VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Modul Marketing pro ERP systém

Marketing module for ERP system

2010

Jan Bauch

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava*.

V Ostravě 16. dubna 2010 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. dubna 2010 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem práce je navrhnout a naimplementovat modul maretingu do stávajícího ERP systému. Modul bude implementován jako webové rozhraní a to pomocí .NET a jako databáze bude použit Microsoft SQL server. Modul musí být uživatelsky pochopitelný a odpovídat požadavkům klientů na zobrazení dat převážně ekonomického charakteru.

Klíčová slova: ERP, marketing, analytiky, datová kostka, datový sklad, .NET, LINQ, SQL, SQL Server Analysis Services, Business Intelligence

Abstract

The goal is to design and implement a marketing module into an existing ERP system. The module will be implemented as a web interface through .NET and as a database Microsoft SQL Server will be used. The module must be user-friendly and fulfil client requirements displaying mainly economic data.

Keywords: ERP, Marketing, Analysis, Data Cube, Data Warehouse, .NET, LINQ, SQL, SQL Server Analysis Services, Business Intelligence

Seznam použitých zkratek a symbolů

ERP	–	Enterprise resource planning
OLAP	–	Online Analytical Processing
OLTP	–	Online transaction processing
DWH	–	Data warehouse - datový sklad
MOLAP	–	Multidimensional Online Analytical Processing
ROLAP	–	Relational Online Analytical Processing
HOLAP	–	Hybrid Online Analytical Processing
MDX	–	Multidimensional Expressions
LINQ	–	Language Integrated Query
SQL	–	Structured Query Language
BI	–	Business Intelligence
SSAS	–	SQL Server Analysis Services
SSRS	–	SQL Server Reporting Services

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Úkolem této práce je navrhnout fungující modul pro již existující ERP systém. Modul musí umět generovat ekonomické kalkulace na základě uživatelových požadavků. Modul bude implementován do stávajícího ERP systému. Tím vznikají určitá technologická omezení a požadavky. Modul musí být implementován pomocí technologie .NET a obsahovat webové rozhraní implementovatelné do stávajícího ERP systému. Při tvorbě webového rozhraní můžeme využít placené komponenty firmy Telerik. Jako databázi bude nutno použít Microsoft SQL Server. Licenci má firma XEVOS Solutions s.r.o. zakoupena v nejvyšší verzi, tudíž je možnost využít veškeré technologie a možnosti, které server nabízí. Vzhledem k nasazení ERP pro firemní klientelu, stačí podpora prohlížečů FireFox verze 3.5 a Internet Explorer verze 8.

V této práci postupně vysvětlím analýzu problému. Uvedeme, proč byla zvolena daná technologie a nevyužity jiné přístupy k vyřešení zadání. Technologie, které v analýze problému vybereme, budou popsány v teoretické části. V implementační části pak vysvětlíme, jak se práce zrealizovala a ukážeme zde provedené postupy při její tvorbě. V závěru práce pak shrneme její výsledek a pokusíme se ohodnotit splnění zadání.

2 Analýza požadavků

Cílem práce bylo vytvořit dle zadání firmy rozšíření jejich stavájícího ERP systému o modul pro marketing. Modul měl umožňovat dolování dat z již existujícího systému dle žádostí klientů. Existující systém byl však velmi rozsáhlý. Databáze obsahovala přes sto tabulek, množství dat u již nasazeného systému bylo ve stovkách MB. Získávání dat mělo v podstatě znamenat vypočítání určitých hodnot a jejich promítnutí v čase, případně podle předem stanovených pravidel. Příklady takových požadavků:

- počet kusů na zboží na skladě,
- hodnota zboží na daném skladě,
- výdělek prodejem daného zboží (ziskovost) a za dané období,
- skupiny cen - kolik typů zboží je v předem definované cenové skupině,
- ...a další.

Navíc některé z uvedených vypočítaných hodnot bylo třeba vyjádřit v procentech. Původně bylo v plánu použít systém na generování SQL dotazů. Nicméně vzhledem ke komplexnosti systému, a tím i složitosti jeho databáze, by takový generátor byl náročný jak na vývoj, tak na případné rozšíření. Dalo se předpokládat, že nároky uživatelů na modul se budou rozšiřovat a pro každý takový požadavek by bylo třeba připravit nový dotaz a zařadit jej do generátoru. Dotaz pro zjištění takových údajů by musel procházet přes několik tabulek. Správa propojení pomocí cizích klíčů by byla náročná na vypracování, tak později by dotaz potřeboval větší množství času i na provedení. Při deseti tisících řádcích a spojení přes několik tabulek by byly nároky na hardware příliš velké. Nehledě na to, že databáze se stále rozšiřuje společně s aktivitami firmy (prodeje, nákupy apod) a například počítat historické výpočty po deseti letech fungování firmy by bylo velmi náročné a zdlouhavé. Dalším důvodem proti SQL generátoru je stále vyvíjející se systém, a s tím i spojený vývoj databáze. Případné úpravy v databázi by se musely ručně upravit v generátoru.

Vzhledem k tomu, že ERP systém běží na SQL serverech od firmy Microsoft a zakoupená licence je dostatečně vysoká, bylo vhodnější použít již existující řešení, a to Business Intelligence (BI) [3] - konktétně pak SQL Server Analysis Services [9] a jeho datové sklady-/kostky. Z modulu pro marketing se tak nakonec stává modul pro komplexní analytiky. Výhodou tohoto řešení je hlavně použití již vytvořené technologie, a to přímo pro tyto účely. Správa datové kostky je jednodušší, a to pomocí grafického rozhraní ve Visual Studiu. Náročnost vytvoření datové kostky je srovnatelná s náročností vytvoření SQL generátoru co do času - rozdílem je použití jiné (nové) technologie. Vzhledem k tomu, že technologie je přímo určená pro dolování dat z velkých databází a jejich agregace, rychlost takového řešení je vyšší při menších nárocích na hardware. I v případě zvýšených nároků by se slabší výkon přenášel pouze na výpočty nad datovou kostkou a nikoliv na primární databázi ERP systému - jinými slovy by vyšší zátěž modulu nezpomalovala práci ostatních uživatelů a celého ERP systému obecně. Datová kostka je pouze prvním

krokem ke splnění zadání. Data z kostky je třeba uživatelům zobrazit. První možností je využít Excel, který je součástí Microsoft Office. Nicméně takové zobrazení není pro uživatele přehledné a celkově práce není tak pohodlná kvůli nutnosti přihlášení přímo k datové kostce (nikoliv k ERP systému a jeho modulu).

K tomu jsem vybral použití technologie LINQ, která přišla jako novinka v nejnovější verzí .NET 3.5 (verze 4.0 v době vývoje modulu nebyla k dispozici). Byla zde možnost použít i reporty z SQL Server Reporting Services. Ty jsou ale vhodnější pro generování statických výstupů - například faktury. I zde je možnost použití grafického rozhraní pro urychlení práce, nicméně opět spíše staticky. Tento způsob vizualizace dat je použit v jiných modulech ERP, bohužel pro tento modul se nezdal být nevhodný. Technologii LINQ byla propojena pomocí ADOMT.NET s datovou kostkou, a bylo tak vytvořeno základní webové rozhraní. První částí webového rozhraní je uživatelský výběr hodnoty k zobrazení a její následná agregace. Možnosti výběru se načítají ze samostatné databáze, tím je zajištěna snadná rozšiřitelnost v případě inviduálních přání klientů. Jako výstup bylo vybráno základní zobrazení. Tím je tabulka a případně volitelně graf.

Použití SQL Server Analysis Services společně s LINQ bylo schváleno firmou. Dále byla možnost využít již zaplacených komponent od firmy Telerik [11]. Vzhledem k tomu, že celé ERP na těchto komponentách funguje (případně se do nich předělává), bylo taktéž využito jejich možností.

3 LINQ - novinka v .NET verze 3.5

S příchodem .NET Framework 3.5 [7] přišel na scénu nový způsob práce s daty na platformě .NET. Do finální verze se dostal projekt LINQ [6], což je jazyk pro dotazování nad jakýmikoliv daty. LINQ - celým názvem Language INtegrated Query - v překladu pak integrovaný jazyk pro dotazování.

Od svého počátku bylo možné v aplikacích postavených nad platformou .NET pracovat s relačními či hierarchickými daty, a to hned několika způsoby. V případě dat, která byla uložena v relační databázi, bylo možné k datům přistupovat skrze rozhraní ADO .NET. Možnost přístupu byla v odpojeném či připojeném režimu. K datům uloženým v dnešní době hodně používaném formátu XML bylo možno využít assembly System.XML. Data pak bylo možno zpracovat sekvenčně nebo jako DOM objekt.

LINQ je jazyk integrovaný přímo do C# nebo VB.NET. Díky integraci odhalíme případné chyby již v době kompilace. Dotazy budou kontrolovány hned, proto se většina chyb objeví před během aplikace. LINQ řeší přístup k neobjektovým datům (XML, relační databáze) v objektových jazycích (C#, VB). Důležitou vlastností je, že tento přístup je řešen bez závislosti na konkrétní technologii. To znamená, že architektura technologie LINQ je navržena tak, že je možné tvořit její implementace pro jednotlivé datové zdroje. LINQ se snaží o urychlení vývoje aplikací - jejich zlevnění - a to tím, že se snaží programátorům maximálně usnadnit práci. Programátor pak nemusí vůbec řešit, kde a jak jsou data uložena. Nemusí psát dotazy "na míru" pro datový zdroj, pouze pomocí LINQ operátorů vytvoří dotaz, který LINQ samostatně namapuje (přeloží) do jazyka zdroje (např. do SQL jazyka). Čili víc říkáme, co má program udělat (zjisti příjmení všech uživatelů) a méně říkáme, jak to má program udělat (SELECT příjmení FROM...). Obecně se dá říct, že umíme-li vytvořit dotaz v LINQ, umíme zároveň vytvořit dotazy pro několik různých jazyků.

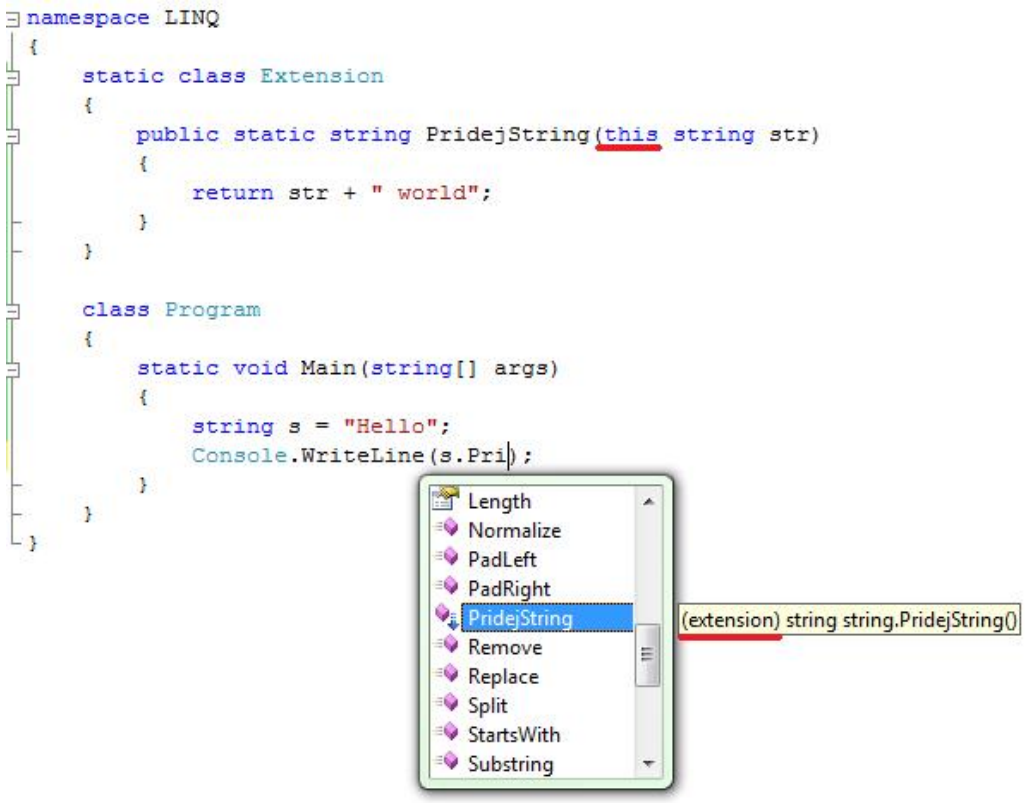
3.1 Novinky v C# použité v LINQ

LINQ využívá několik novinek z .NET Framework verze 3.5. Není nutné popisovat všechny novinky, proto zde naznačím nové funkce jazyka týkající se přímo jazyka LINQ.

3.1.1 Extension metody

Tato nová vlastnost rozšiřuje již existující typ o nové instanční metody, a to bez toho, abychom museli jakýmkoli způsobem do daného typu zasáhnout. Právě rozšíření stávajících typů přineslo mnohem pohodlnější práci s daty a zjednodušení zápisu v kódu. Extension metody jsou implementovány jako speciální druh statických metod. Aby mohla být extension metoda použita, musí být typ, který ji hodlá využít ve stejném jmenném prostoru jako třída, jež danou extension metodu obsahuje. Případně je třeba tento jmenný prostor importovat.

Jak můžeme vidět na obrázku 1, Extension metody se vyznačují použitím klíčového slova *this* na první vstupní argument metody. Visual Studio 2008 [10] pak tyto metody umí nabídnout pomocí IntelliSense, navíc tyto metody rozpozná a označí pomocí popisku



Obrázek 1: Extension metody

(*extension*). Program na obrázku představuje ”Hello world” ukázku. Metoda *PridejString* připojí string *world* text k původnímu *Hello*. Program tedy vypíše *Hello world*.

Extension metody se používají zejména k přidávání funkcionality k rozhraním, případně k doplnění metod k existujícím .NET typům. Nezvyklou vlastností je chování při ”null” hodnotě - viz. ukázka 1. Kompilátor v tomto případě nevrací NullReferenceException. V tomto případě se metoda normálně zavolá, akorát bude její první argument s klíčovým slovem this roven hodnotě ”null”.

```
string str = null;
// zadna vyjimka – novyString bude hodnoty "World"
string novyString = str.PridejString("World");
```

Výpis 1: Ukázka Extension metod

3.1.2 Anonymní typy

Tato novinka umožňuje definovat nové typy za běhu a rovnou vytvořit jejich instanci. Stejně jako předchozí novinka i tato zjednodušuje zápis kódu. Anonymní datové typy v LINQu nelze použít jako návratový typ z metody a jediný způsob, jak anonymní

typ z metody předat, je použít jako návratovou hodnotu typ Object -viz. 2. Anonymní typ je vždy deklarován jako *internal* a jeho použití je tak striktně omezeno na jednu assembly. Kompilátor vytváří anonymní typ při kompilaci, nikoliv za běhu programu. V době kompilace kompilátor vytvoří nový anonymní typ. Nový typ v následujícím příkladu bude obsahovat vlastnosti *Jmeno* a *Prijmeni*. *Get* a *Set* metody jsou generovány automaticky.

```
var osoba = new {Jmeno = "Petr", Prijmeni = "Novák"};
//Vypise: Petr Novak
Console.WriteLine("{0} - {1}", osoba.Jmeno, osoba.Prijmeni);
```

Výpis 2: Anonymní typy - Object

Pokud je programátor zvyklý na jiné jazyky (např.: PHP), mohl by si myslet, že s takto deklarovanou proměnnou může dělat cokoliv. Jak jde vidět na následujícím příkladu 3, toto v .NET neplatí. Klíčové slovo určí datový typ podle pravé strany výrazu za rovná se. Podle toho určí datový typ proměnné.

```
var x = 1;   // v tomto příkladu je tento zápis shodný se zápisem: int x = 1;
x = "1";  // vypíše chybu – není možno konvertovat string na int
```

Výpis 3: Anonymní typy - datový typ

Jak dále uvidíme, tato novinka se využívá v podstatě v každém LINQ příkazu.

3.1.3 Lambda výrazy

Tato novinka byla převzata z funkcionálního programování. Opět se jedná o novinku hojně využívanou v LINQ. A stejně jako předchozí novinky, i tato slouží především k zjednodušení zápisu kódu. Jak dále uvidíme, zjednodušení je opravdu znatelné. Ve verzi C# 2.0 byla představena nová vlastnost - anonymní metody. Ty umožňovaly deklarovat metody na řádku. Lambda výraz používá lepší syntaxi k dosažení stejného cíle. S lambda výrazy přichází do jazyku C# nový operátor =>, který je nazýván "přechází v".

Obecný tvar lambda výrazu: (vstupní argumenty) => výraz

Příklad:

```
(int x) => x + 1          // explicitní parametr = je určen datový typ vstupního argumentu
(y,z) => return y * z;    // implicitní parametr = není určen datový typ vstupního argumentu
```

Výpis 4: Lambda výrazy - obecný tvar

Příklad - C# 2.0 oproti C# 3.0:

```
List<int> cisla = new List<int> { 1, 2, 3, 4, 5 };
// Kód v C# 2.0
List<int> result = cisla.FindAll(delegate(int i)
{
    return i < 4;
});
// to samé zjednodušeně v C# 3.0 pomocí lambda výrazu:
List<int> result = cisla.FindAll(i => i < 4);
```

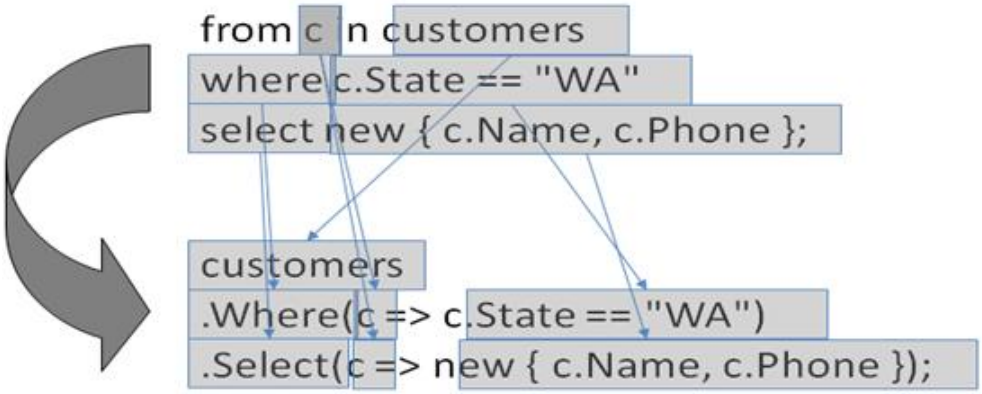
Výpis 5: Lambda výrazy - porovnání verzí C#

3.1.4 Výrazové stromy

Tato novinka umožňuje pracovat s kódem nikterak jako se spustitelnými příkazy, ale jako s daty. S její pomocí je možné v aplikaci vytvořit stromovou strukturu představující kód. S touto strukturou pak může pracovat jiný kód. Případně může data zkompilovat pomocí přikazu *Compile()* a kód spustit. Toho využívá právě LINQ. Výrazové stromy (expression trees) v případě LINQ reprezentují strukturu dotazů pro cílová data, které implementují IQueryable(T). Pokud budeme chtít vytvořit vlastní provider pro LINQ, budeme v podstatě vytvářet jen IQueryable(T) interface, který bude dotazovat cílový zdroj dat.

3.1.5 Query Expression

Query Expression jsou pro vývojáře nejviditelnější části LINQ. Jedná se o dotazy, které vytvoří sám programátor. Query expression píšeme v deklarativní syntaxi podobné SQL. Tento zápis nám umožňuje operace nad daty - lehce tímto způsobem můžeme data filtrovat, spojovat, řadit apod. Pomocí query expression na data z kterékoliv podporovaného datového zdroje. Například můžeme načíst data z XML dokumentu a uložit je do MSSQL databáze. Query Expression jsou přímo součástí C# jazyka, proto chyby odhalíme už při překladu. V době překladu se dotaz překládá do volání metod - viz. obrázek 2:



Obrázek 2: LINQ - Query Expression [14]

Příklad přeložení dotazu z LINQ do SQL. Programátor napíše dotaz v LINQ pomocí dotazu (query syntax):

```
var q = from c in db.Zákazník
        where c.Město == "Ostrava"
        select c;
```

Výpis 6: Query Expression - query syntax

nebo pomocí volání metod (method syntax):

```
var q = db. Zákazník
        .Where(c => c. Město == "Ostrava")
        .Select(c => c);
```

Výpis 7: Query Expression - method syntax

Oba tyto zápisy jsou ekvivalentní. Doporučuje se používat první zápis, hlavně kvůli přehlednosti. Když je dotaz kompilován, je vždy přeložen do zápisu pomocí metod, protože .NET Common Language Runtime (CLR) zvládá pouze zápisu pomocí volání metod.

A tento zápis je přepsán pomocí provideru na SQL dotaz:

```
SELECT c
FROM db. Zákazník AS c
WHERE c. Město = 'Ostrava'
```

Výpis 8: Query Expression - SQL

Způsoby provedení dotazu:

- Odložené

  Jak můžeme vidět na obrázku 3, dotaz se stardardně provede, až po použití ve foreach funkci. To umožňuje vytvořit sadu dotazů předem a jejich vykonání provést až v případě potřeby. Následující dotaz 9 vyhledá uživatele s platem vyšším než 10000.

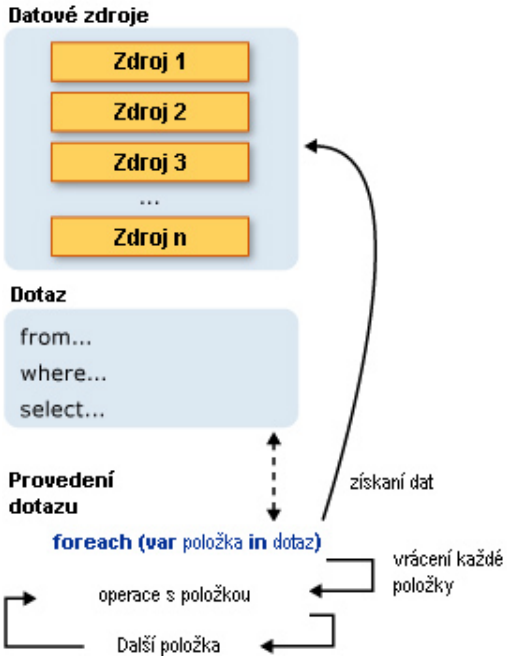
```
// programátor vytvoří dotaz
    var query = from item in db.Users
                where item.plat > 10000
                orderby c.plat descending
                select item;
// dotaz se provede až v tomto okamžiku
foreach (var item in query)
{
Console.WriteLine("Nalezen uživatel " + item.jmeno + " s platem " + item.plat);
}
```

Výpis 9: Query Expression - odložené provedení dotazu

- Okamžité

  Dotazy (viz. 10), které používají agregované funkce, lze provést okamžitě. Jedná se například o funkce *Count*, *Max*, *Average* nebo *First*. Tyto dotazy se v kódu provádí bez foreach. Tyto dotazy vrací pouze jednu hodnotu (v případě príkladu integer), nikoliv IEnumerable kolekci.

```
var query =
from num in numbers
    where (num \% 2) == 0
    select num;
```



Obrázek 3: Odložené provedení LINQ dotazu

```
int query = query.Count();
```

Výpis 10: Query Expression - okamžité provedení dotazu

Tento dotaz vrátí počet řádků, které odpovídají podmínce dotazu. Funkce *Count()* obsahuje foreach v těle, proto není nutné použít foreach přímo v těle kódu.

3.2 LINQ syntaxe

Jak bylo řečeno výše, syntaxe LINQ je podobná SQL. LINQ přebírá od SQL klíčová slova. Tím usnadňuje jak tvoření dotazů, tak pochopení jejich významu. Pokud programátor zná SQL, nebude pro něj problém číst LINQ prakticky okamžitě. Tvorba dotazů pro osoby znalé SQL je pak otázka několika málo pokusů.

Základem tvorby dotazů jsou klíčová slova:

- From - určuje datový zdroj,
- Select - výběr hodnoty kterou chceme použít,
- SelectMany - výběr více hodnot najednou (např. pole),
- Join - spojení více poskytovatelů dat,
- GroupBy - rozdělení dat do více skupin podle určitého klíče,

- Where - omezení výběru prvků podle specifikované podmínky,
- OrderBy, OrderByDescending - specifikace třídění,
- First, Last - výběr prvního nebo posledního prvku z kolekce,
- ElementAt - výběr prvku podle udaného indexu,
- Count - počet prvků v kolekci,
- Union, Intersect, Except - definice množinových operací sjednocení, rozdíl a průnik,
- Sum, Min, Max, Average - vrací součet, minimální, maximální či průměrnou hodnotu z dané kolekce,
- Reverse - otočí pořadí prvků v kolekci,
- Concat - spojí dvě kolekce dohromady,
- OfType - výběr pouze těch prvků, které jsou specifikovaného typu.

Základní syntaxe LINQ dotazů pak je:

```
from [typ] proměnná in datový_zdroj
[where] podmínka_restrikce
[orderby] klíč_řazení [ascending | descending]
select výraz_projekce
```

Výpis 11: LINQ syntaxe

Příklad s podmínkou - WHERE a řazením od nejvyššího platu

```
var q =
from c in db.Uživatelé
    where c.plat > 10000
    orderby c.plat descending
select c;
```

Výpis 12: LINQ syntaxe

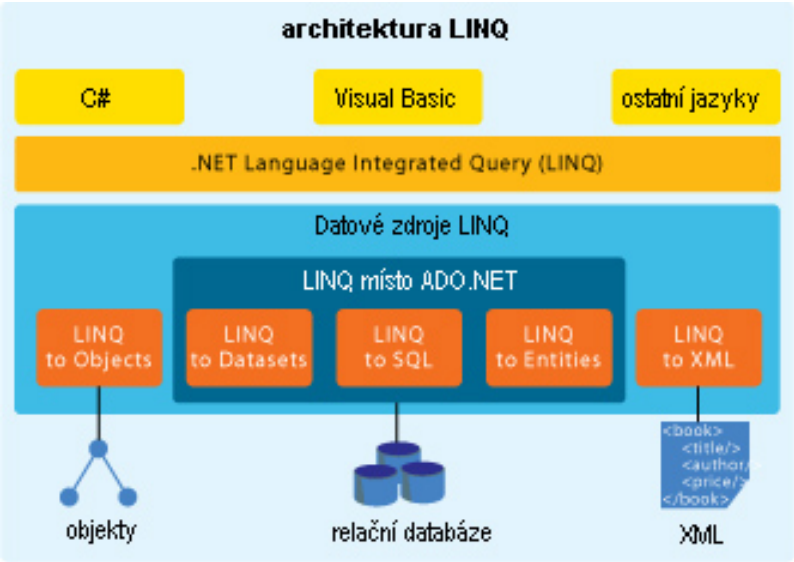
3.3 LINQ-to-kdeco

LINQ je navrhnut jako poměrně obecný nástroj, takže je možné v něm manipulovat s různými daty. Čtyři základní zdroje dat, se kterými LINQ umí pracovat, jsou:

- LINQ to Objects - umožňuje dotazování nad objekty (respektive jejich kolekcemi). Implementace určena pouze pro práci s daty, které se nachází v paměti.
- LINQ to XML - umožňuje pracovat s XML soubory. Plně objektový přístup k datům.
- LINQ to DataSet - Implementace LINQ pro práci s ADO .NET datasety

- LINQ to SQL - popsáno dále
- LINQ to cokoliv - vznikají další a další implementace. Například LINQ to Amazon - pro vyhledávání titulů na amazon.com, LINQ to MySQL - pro dotazování nad MySQL databází atd.

Na obrázku 4 lze vidět znázorněnou architekturu včetně zdrojů dat:



Obrázek 4: Architektura LINQ

3.4 SQL

Lidé mají potřebu shromažďovat informace už odpradávna. V dnešní době se bez evidování prakticky neobejdeme - evidence občanů, aut, hospodářství. . . V dnešní moderní době počítačů a výpočetních center chceme informace ukládat hlavně v digitální podobě, vyhledávat v nich, třídit je apod. K tomuto účelu používáme databáze. Ale jak s databází komunikovat? Jak si říct, která data chceme vyhledat? K tomuto účelu vznikl procedurální standardizovaný jazyk SQL. SQL celým názvem Structured Query Language, česky pak strukturovaný dotazovací jazyk, který je určen pro práci s daty v relačních databázích.

3.4.1 Historie

Výzkum relačních databází probíhal již v 70. letech 20. století ve firmě IBM. Bylo nutno vytvořit sadu příkazů pro ovládání těchto databází. V roce 1974 IBM vydalo jazyk Sequel (Structured English Query Language, česky pak anglický strukturovaný dotazovací jazyk). Cílem bylo vytvořit jazyk, ve kterém by se příkazy tvořily syntakticky co nejblíže přirozenému jazyku (angličtině). K vývoji se později připojí několik společností včetně

dnes známého Oracle. Jazyk byl nakonec přejmenován na SQL. V roce 1986 byl jazyk SQL prohlášen standardem ANSI (SQL 86), o rok později schválen standardem ISO (SQL 87). Jazyk SQL se dnes používá prakticky ve všech relačně-databázových systémech, i když většinou mírně upravený. Obvykle se dodržují základní příkazy SQL a systémy se rozšiřují o vlastnosti, které nejsou popsány v ANSI. Přenositelnost mezi databázovými systémy existuje, ale pouze v omezené míře. Příkladem databázových systému dnešní době jsou populární MySQL používaný nejčastěji společně s programovacím jazykem PHP pro webové aplikace, velmi robustní (a drahý) Oracle, dále například MSSQL, se kterým právě pracuje LINQ to SQL.

3.4.2 Popis SQL

Jazyk SQL [2, 8] je tvořen pomocí klíčových slov a podobá se angličtině. Díky tomu je pochopení základů jazyka snadné. Jednoduchost by však nebyla jediný důvod proč zrovna SQL. SQL zvládá chráněný síťový přístup - přístup více klientů v jednom čase. Jedná se o zámky případně novější transakce. Jazyk SQL umožňuje plnou kontrolu nad databázovým systémem. Jedná se o:

- získávání/vyhledávání dat,
- přidávání/změna/mazání dat,
- vytváření databází a tabulek v databázích,
- vytváření uložených procedur a pohledů,
- nastavení práv na tabulky, databáze, procedury, pohledy.

Podle těchto vlastností dělíme SQL příkazy do čtyř skupin:

- **Příkazy pro definici dat** - (DDL - Data Definition Language) - těmito příkazy vytváříme struktury databáze. Vytvořené skupiny lze upravovat a mazat. Jedná se například o tvorbu tabulek, práce s indexy, pohledy apod. Příklady příkazů:
  - CREATE - vytváření objektů
  - ALTER - změna objektů
  - DROP - mazání objektů
- **Příkazy pro manipulaci s daty** - (DML - Data Manipulation Language) - zde patří příkazy pro manipulaci s daty. Příklady příkazů:
  - SELECT - výběr dat
  - UPDATE - změna dat
  - DELETE - mazání dat
  - INSERT INTO - vkládání dat

- **Příkazy pro řízení dat** - (DCL - Data Control Language) - příkazy pro nastavení přístupových práv a řízení transakcí Příklady příkazů:
  - GRANT - přidělení práv
  - REVOKE - odebrání práv
  - START TRANSACTION - zahájení transakce
  - COMMIT - potvrzení transakce
  - ROLLBACK - zrušení transakce
- **Ostatní příkazy** - zde patří příkazy pro správu uživatelů, nastavování systémových parametrů apod. Tato skupina není standardizovaná a syntaxe příkazů se liší podle databázového systému

Pravděpodobně nejpřínosnější a nejpoužívanější technologie z rodiny LINQ. LINQ to SQL, dříve nazývaný DLINQ, je v podstatě objektově-relační mapper, který je napsán tak, aby šlo s daty manipulovat pomocí základních LINQ funkcí. Jedná se o provider, který nám dovolí provádět běžné databázové operace v MSSQL - a to díky LINQ - silně typově bez nutnosti psát přímo SQL kód přímo pro MSSQL v datové vrstvě. LINQ tedy nahrazuje ADO.NET a datasety. LINQ můžeme psát jak v C# tak ve VB - kód pak bude automaticky přemapován do příslušných MSSQL dotazů, které jsou vykonány přímo v databázi.

3.5 Modelování databáze

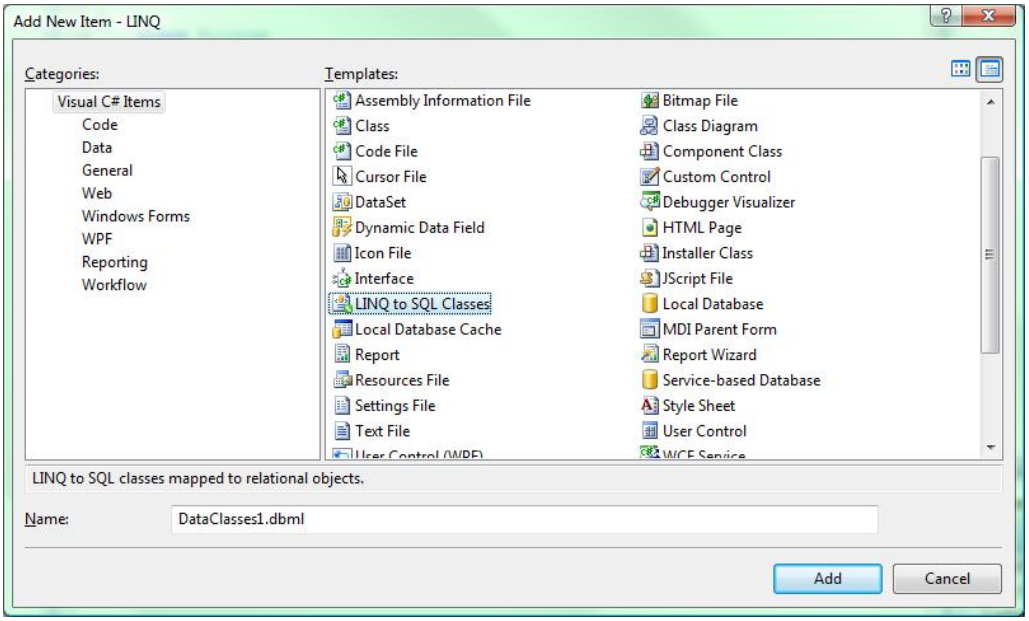
Visual Studio obsahuje nástroj SQL Designer, který umožňuje jednoduše vytvořit vizuální model databáze jako LINQ-SQL objektový model. Model vytvoříme tak, že do projektu přidáme novou položku *LINQ to SQL Classes* - viz obrázek číslo 5 níže:

Třídy (Entity Classes) můžeme kreslit ručně. Designer nabízí celou řadu důležitých funkcí. Vytvoření tříd včetně vazeb mezi nimi. Přidávání/úpravy atributů tříd primární klíče apod. (popsáno dále). Designer pak podle návrhu vytvoří tabulky v databázi.

Druhou možností je vytvořit třídy přímo v kódu - viz ukázka kódu číslo 13. V podstatě je třeba vytvořit mapování mezi LINQ objekty a vzdálenou databází ručně v kódu. Stejně jako v předchozím způsobu, lze vytvořit kompletní vazby, definovat datové typy atributů atd. Ukázka vzorové tabulky zákazníků s atributem id a jménem:

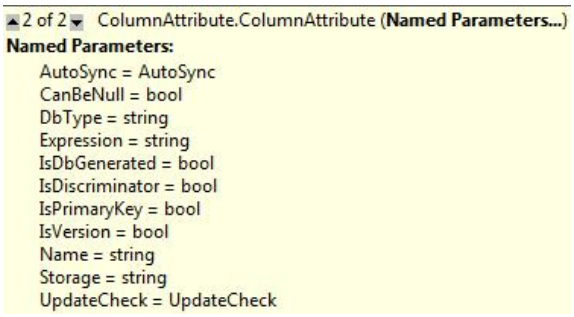
```
[Table(Name = "Zakaznici")]
    public class Zakaznik
    {
        [Column(IsPrimaryKey = true)]
        public int id_z;

        [Column]
        public string jmeno;
    }
```

Výpis 13: Modelování databáze



Obrázek 5: Modelování databáze

Vytvořený objekt odpovídá tabulce v databázi. Proměnné musí odpovídat atributům v tabulce typově i podle názvu. MSSQL nerozlišuje velká a malá písmena, proto název nemusí být case-senzitivní. Asociaci vlastností s atributy v tabulce provádíme pomocí hranatých závorek. Nejprve mapujeme celou tabulku, poté jednotlivé atributy. V případě uvedeného příkladu je atribut *id_z* označen jako primární klíč. Kromě primárního klíče lze nastavit několik dalších vlastností, jejich seznam nám poskytne přímo Visual Studio - viz obrázek 6 :



Obrázek 6: Parametry mapování

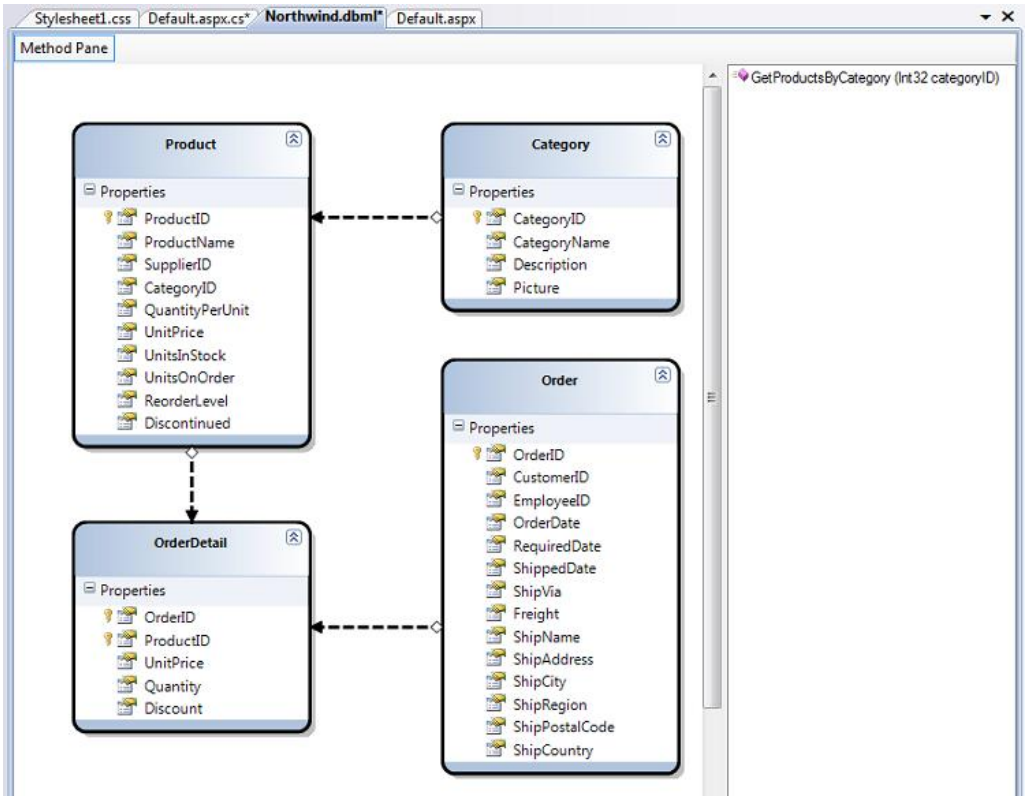
Samotné mapování provádí *DataContext* - viz 14. Ten překládá požadavky z objektů na databázi do SQL a naopak. DataContext používá connection string, pomocí kterého se připojí na server a umožní nám vytvořit Table<T>. T je typ, na který bude databáze mapována. Table<T> zahrnuje data v tabulce a implementuje IQueryable<T> interface.

Tím je vytvořen expression tree pro LINQ to SQL provider. Následuje ukázka vytvoření DataContext, Table<T> a ukázkový dotaz nad touto tabulkou.

```
DataContext databaze = new DataContext(connection_string);
Table<Zakaznik> Zakaznici = databaze.GetTable<Zakaznik>();
var q =
from c in Zakaznici
        where c.jmeno == "Jmeno"
        select c;
```

Výpis 14: Modelování databáze - příklad s podmínkou

Nejrychlejší možností je vytvoření objekty podle tabulek v databázi. Tabulky stačí přetáhnout myší z datových připojení, Designer (obrázek 7) zařídí všechno automaticky. Krom mapování tabulek na LINQ objekty umožňuje i použití uložených procedur na MSSQL serveru.



Obrázek 7: Objekty podle databáze

Na tomto obrázku vidíme návrh databáze vytvořený způsobem drag&drop. Objekty odpovídají tabulkám na serveru. Designer dále načetl atributy včetně jejich typu a dokázal rozlišit primární klíče. Šipky mezi třídami/objekty reprezentují asociace mezi nimi. Asociace jsou obvykle modelovány podle primárních/cizích klíčů v databázi. Vazba může být buď jedna k jedné nebo jedna ku N. Příklad jedna k jedné může být vztah člověk a jeho

právě jedno rodné číslo. Jedna ku N je pak například vztah učitel učí několik předmětů. A nakonec v pravém sloupci se nachází seznam uložených procedur.

3.6 LINQ - Příklady

- **dotazování na data** - výběr dat
  - dotaz vybere všechny akcie podle podmínky *WHERE*

```
var akcie =
from p in produkty
where p.jedotek_dispozici > 0 && p.cena_jednotky > 3.00M
select p
```

Výpis 15: LINQ - Příklady - dotazování

  - seřadí produkty podle jména - klíčově slovo *ORDERBY*

```
var sezarene_produkty =
    from p in produkty
orderby p.jmeno
select p;
```

Výpis 16: LINQ - Příklady - řazení

- **úprava dat**

```
// vybere zaměstnance podle jména
Zamestnanec zam = db.Zamestnanci.Single(p => p.jmeno == "Penny");
// zvýší plat
zam.plat += 500;
// a aktualizuje položku v databázi
databaze.SubmitChanges();
```

Výpis 17: LINQ - Příklady - úprava dat

- **vkládání dat**

  Vkládání dat vyžaduje několik kroků. Nejdříve je třeba vytvořit novou položku a nastavit její atributy. Dále je třeba ji přiřadit k objektu (z pohledu MSSQL přiřazujeme nový vytvořený řádek k tabulce). Posledním krokem pak je aktualizovat změny v databázi. Je třeba si uvědomit, že automaticky generované primární klíče stejně jako cizí klíče nemusíme řešit ručně. LINQ to SQL vytvoří tyto vazby automaticky.

```
// chceme nového zaměstnance
Zamestnanec zam = new Zamestnanec();
// přidáme jméno
zam.jmeno = "Nový";
// zadáme plat
zam.plat = 11000;
// vytvoření vazby nová položka – objekt
databaze.Zamestnanci.Add(zam);
```

```
// aktualizujeme databázi
databaze.SubmitChanges();
```

Výpis 18: LINQ - Příklady - vkládání dat

- **mazání dat**

  Mazání dat je obdobně jednoduché, jako ostatní operace. Probíhá ve třech krocích. Nejdříve vybereme data, která chceme smazat. V našem případě vybíráme všechny zaměstnance s platem nad 100000. Nalezené položky smažeme a nakonec jako obvykle aktualizujeme databázi.

```
// vyhledáme zaměstantce, které chceme smazat
var ke_smazani =
    from c in db.Zamestnanec
    where c.plat > 100000
    select c;
// smažeme
databaze.Zamestnanci.RemoveAll(ke_smazani);
// a aktualizujeme databázi
db.SubmitChanges();
```

Výpis 19: LINQ - Příklady - mazání dat

3.7 Příklady mapování

Jak jsme si ukázali, tvorba dotazů v SQL může být složitá a vyžaduje studování nového jazyka. A právě proto přichází LINQ. Přestože primárním přínosem LINQ je maximální efektivita při dotazování nad daty, u většiny implementací LINQ to u dotazování nekončí a kromě toho je možné data nejen efektivně dotazovat, ale i také modifikovat, propojovat, třídit, vyhledávat apod. K tomu slouží skupina operátorů. A to hlavně jednoduše, v kódu a bez znalosti SQL.

```
var q =
    from c in db.Users
    where c.plat > 10000
    orderby c.plat descending
    select c;
```

Výpis 20: Příklady mapování - LINQ

se mapuje na:

```
SELECT [t0].[id_z], [t0].[jmeno], [t0].[plat]
FROM [dbo].[User] AS [t0]
WHERE [t0].[plat] > @p0
ORDER BY [t0].[plat] DESC',N'@p0 int',@p0=10000
```

Výpis 21: Příklady mapování - SQL

4 Business Intelligence (BI)

V ERP systémech se v současnosti jedná o poměrně často používaný pojem. Business Intelligence [3] zastřešuje všechny techniky pro operace s daty v dané firmě - systému. Zakladatel analytik Howard J. Dresner shrnul v roce 1989 svou představu o Business Intelligence jako "sadu konceptů a metod určených pro zkvalitnění rozhodnutí firmy" s využitím různých funkcí a nástrojů. Termín Business Intelligence (zkráceně BI) se v současnosti používá i v české odborné teorii a praxi. Termín se používá v anglické podobě z důvodu nenalezení výstižného českého překladu. Dalším důvod mohla být mezinárodní platnost tohoto termínu. Nejbližším českým překladem by mohl být termín "obchodní inteligence", nicméně se v praxi ani literatuře běžně nepoužívá. Důvod vzniku BI byl ovlivněn rozvojem oblastí jako informatika, ekonomika a velkým nárůstem dat a informací. V dnešní době má každá větší společnost/firma uloženo velké množství obchodních dat, které je třeba zpracovat, analyzovat, třídit apod. Společně s nárůstem dat vznikl termín "informační potřeba". Ve zkratce se jedná o potřebu informovat okolí a sebe. Tato potřeba se vyskytuje na začátku každého hledání informací. Ať se jedná o potřebu vyhledat nejbližšího lékaře nebo čistě obchodní záležitosti typu zjištění pěti nejprodávanějších druhů zboží v daném čase a konkrétní oblasti prodeje. Zápornou stránkou informační potřeby je, že s nárůstem objemu dat vzniká tzv. informační přehlcení/přetížení. V takové situaci lze těžko (nebo vůbec) získat relevantní informace. Případně uživatel vůbec neví, zda konkrétní informace existuje nebo kde ji hledat. Krom nárůstu objemu dat mohou být příčinou přehlcení nekonzistentní data, chyby aplikací apod.

Na začátku směřovala BI hlavně k vedoucím pracovníkům - top managementu. Jak se BI vyvíjela, došlo k rozšíření na větší počet uživatelů systému. Tento vývoj umožnil lepší prodej systému - firma nekupovala systém pro několik málo zaměstnanců, ale téměř pro každého. Cílem aplikace BI je tedy dostupnost potřebných dat všem uživatelům systému. Každý uživatel potřebuje mít dostupné pro něj důležité údaje. Pro manažera to může být vývoj cen za poslední rok, pro skladníka procentuální obsazení skladů apod. Díky analýze a prostředkům BI lze tyto data zpracovat a získat a tím zrychlit procesy ve firmě. Business Intelligence dává správné informace správným lidem ve správný čas, aby jim pomohla dělat správná rozhodnutí k získání konkurenční výhody. Z tohoto plynou další požadavky - například doručení informací. Každý uživatel chce informace získat jiným způsobem - Excel, web, mobil, email apod. Tím ovšem narůstá i cena projektu a je třeba zvážit, jestli se investice do takového systému vyplatí.

Z manažerského hlediska jde v případě BI o prostředek pro využití dat firmy (prodejnost v regionech, ceny nákupu zboží apod.) pro analýzu dat a z toho plynoucí lepší rozhodování v budoucnosti a přehledné zobrazení dat z minulosti. BI v tomto případě umožní efektivní zpracování dat dané firmy. Z informačního hlediska je business Intelligence vnímána spíše jako technické zajištění realizace. Realizace BI je tedy založena na práci s daty, která jsou zpracována ve srozumitelné formě uživatelům. Data mohou být strukturovaná - bývají uložena v relačních databázích, přímo obsahují fakta, hodnoty apod. Dalším typem jsou data nestrukturovaná - například obrázky, videa apod. BI se obvykle zabývá prvním typem. Manipulace, ukládání a použití tak velkého množství dat je v tomto případě možné jen díky výpočetní technice. Business Intelligence využívá mo-

derní a specifické prostředky z oblasti IT, např.: reportování (zobrazení konkrétních dat uživateli), analytiky (jaká data uživatel chce), datové sklady, datová tržiště, data mining (dolování dat) apod.

Díky tomu lze získat potřebné informace. Data samy o sobě nemají žádný význam. Databáze pouze ukládají nějaké hodnoty. Abychom dostali potřebné informace, je třeba data vhodně zpracovat. A právě úspěšné zpracování dat má rozhodující vliv na kvalitu informacích pro uživatele v systému. Čili informace jsou výsledkem zpracování dat a slouží pro uspokojení informační potřeby uživatele. Aby byla informace správná, potřebujeme splnit mimo jiné tyto vlastnosti:

- aktuálnost,
- úplnost,
- relevantnost (jestli je hledaná informace správná v konkrétním kontextu),
- pravdivost.

Dalším krokem systému je pak schopnost se získanou informací správně naložit. Informace musí být srozumitelné a mít přehledné zobrazení, tabulka případně graf. Informace se musí k uživateli dostat včas v situaci, kdy je uživatel potřebuje. Nakonec je třeba zajistit, aby systém předal informaci správnému uživateli, tzv. kompetentnost.

4.1 Vrstvy Business Intelligence

Business Intelligence je komplexní prostředek a tomu odpovídá i jeho komplexnost. BI obsahují pět vrstev a při návrhu systému se u každé předpokládá její použití, proto jsou zde popsány podrobněji.

- Zdroje dat
- Transformace dat
- Ukládání dat
- Analýza dat
- Prezentace dat

4.1.1 Zdroje dat

Primární (provozní) systémy tzn. systémy, které obhospodařují požadavky uživatelů v reálném čase, jsou obvykle jediným zdrojem dat pro BI. Kvalita, konzistence a množství těchto dat jsou základem pro aplikaci BI. Tyto systémy lze označit také anglickou zkratkou OLTP - (On-line Transactional Processing) a jsou určeny pro ukládání a získávání dat ve víceuživatelské prostředí v reálném čase. Takové databáze neobsahují redundance a

jejich tabulky měly by být v nejméně třetí normální formě. Pak se jedná o tzv. normalizovanou databázi. Normalizace se provádí z důvodů odstranění redundancí. Tím se zajistí, že nebude docházet k nekonzistenci dat, případně jiným chybám při manipulaci s daty. Provádění složitých a náročných analýz dat v takovém prostředí by vedlo k přetížení a nezvládnutí primárních úkolů. Management firmy obvykle generuje jiné dotazy než běžní uživatelé - nezajímají je ani tak jednotlivé záznamy, ale spíše celkové informace. Management tedy požaduje výsledky z tzv. intenzivních dotazů. Výsledky nejsou v databázích uloženy přímo a je nutné je dlouze počítat. Před aplikací BI systému jsou primární databáze také jediným zdrojem dat pro uživatele. V praxi se nejčastěji jedná o relační databáze a případně data z externích zdrojů (webové služby apod.). A právě odlišnost datových zdrojů je problém při jejich analýze nebo vytváření reportů, tiskových sestav apod. Úkolem BI je tedy data analyzovat z pohledu firemních požadavků a pro potřeby řízení firmy.

4.1.2 Transformace dat

Tato vrstva slouží k načtení dat ze všech zdrojů, jejich úpravě a nahrání do datových skladů. Tato vrstva se taky označuje jako ETL (Extraction, Transformation and Loading). Extraction ve smyslu získání dat ze zdrojů, transformation znamená upravení dat do požadované formy a nakonec Loading je import již upravených dat do datových skladů.

- **Extraction** - zde probíhá výběr atributů, které bude třeba do datového skladu načíst. K tomuto kroku tedy potřebujeme znát schéma zdrojových databází - případně dalších zdrojů. Pak můžeme vybrat důležité tabulky a jejich atributy. Atributy je třeba projít a vybrat, které vypustíme a označit ty, které je třeba přepočítat - např. cena v jednom datovém zdroji může být v korunách a v jiném v eurech, dolarech atd. Jedná se o náročnou a velmi důležitou část návrhu datových skladů. Je zde třeba spolupráce s uživateli systému - každý uživatel ví nejlépe, která data chce a každý uživatel chce vidět jiná data. Dále je třeba některá data dopočítat - k tomu lze použít například pohledy (views) v databázích apod.
- **Transformation** - zde se uplatní tzv. datová pumpa. Jejím úkolem je již vybraná data zpracovat. Čili část dat z provozních systémů a ostatních datových zdrojů převést do formátu pro datový sklad. Tento proces občas představuje změnu dat (viz přepočet cen zmíněný výše) a taky jejich filtraci. Datová pumpa "čistí" (kontroluje) správnost dat. Zde patří například kontrola duplicitních údajů. Takové údaje je třeba smazat a měl by se doporučit kontrolu původního systému - duplicitní údaje by se v provozním systému nemají vyskytovat. Oprava chyb není třeba, pokud jsou data ve zdrojové databázi v pořádku. Případnou chybu lze opravit v datové pumpě nebo aplikačně v původním systému případně přímo ve zdrojové databázi. Dalším problémem mohou být chybějící hodnoty. Ty lze doplnit z jiného zdroje, případně je do datového skladu vůbec nezařadit. V této části sjednocujeme zápisy atributů z různých zdrojů. PSČ může být ve formátu XXX XX, jinde XXXXX, obdobně datum narození apod. Problém toho typu se často vyskytuje ve velkých firmách, kde bývají číselníky nejasné - například různá oddělení mají různá čísla pro stejné zboží

apod. V praxi je datová pumpa složena z několika programů, které jsou přizpůsobeny zdrojovým databázím a datovému skladu. Datová pumpa krom čištění dat obstarává i výpočet navržených agregací. Agregací nazýváme hodnoty běžných agregačních funkcí z SQL - například, SUM (součet), AVG (průměr) apod.

- **Loading** - zde se provádí přenos dat z primárních systémů do datového skladu. Při vytvoření datového skladu se najednou nahrají všechna data. Loading pak pokračuje průběžně podle potřeby, kdy se přenášejí nová data přidaná do systému po provedení prvního načtení dat. Takové načítání se obvykle provádí automaticky v určitých časových intervalech.

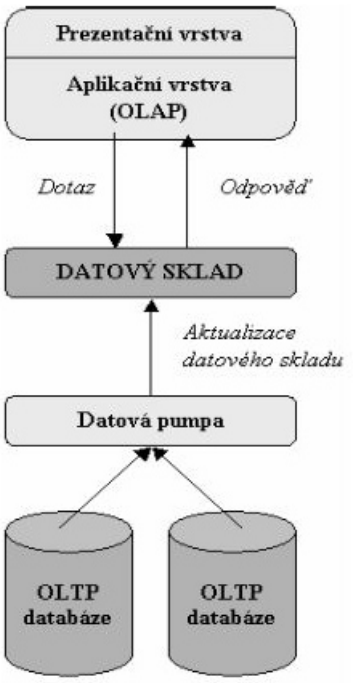
4.1.3 Ukládání dat

Data jsou ve společnostech uložena na více místech a v různých systémech. Protože tyto systémy mezi sebou obvykle nekomunikují - nebo jen velmi omezeně, nejsou schopny vrátit uživateli relevantní informace. Abychom mohli získat požadovaná (agregovaná) data, je třeba data prvně uložit do vhodných struktur. K tomuto účelu se používá tzv. datový sklad.

4.1.3.1 Datový sklad (DWH - data warehouse) [5] Jedná se o databázi obsahující upravená data (viz datová pumpa výše) ze všech provozních systémů. Datový sklad bývá optimalizován pro reporting, analýzu a archivaci dat (nikoliv pro rychlé zpracování transakčních operací). Z toho důvodu se zde mohou objevit duplicitní záznamy, jejíž výskyt je v klasických relačních databázích nežádoucí. Přepočítávání agregací při každém dotazu z atomických dat v transakčních databázích je hardwarově náročné a proto se v datových skladech obvykle i za cenu vetší spotřeby paměti agregace předpočítávájí. Agregace se ukládají v tzv. multidimenzionálním tvaru.

Na obrázku 8 můžeme vidět základní strukturu datového skladu. Datová pumpa zajišťuje aktualizace z primárních systémů. Nad datovým skladem se nachází aplikační vrstva OLAP (bude vysvětleno dále), která se dotazuje datového skladu na data. Výsledek je pak promítnut do prezentační vrstvy, která se stará o samotné zobrazení uživateli (tabulka, graf apod.) Nároky na hardware se zde přesouvají do transformační vrstvy, kde se výpočty provedou a do datového skladu se uloží již přepočítané hodnoty. Při dotazech se tedy agregace již nepočítají, pouze se vyhledají v datovém skladu. Redundance zde urychlují vyhledávání a u datových skladů se v praxi tento způsob ukládání používá nejčastěji. Datový sklad tedy není na rozdíl od transakčních databází v 3. normálové formě.

4.1.3.2 Data Warehousing Jedná se proces (často bývá zaměňován za produkt), který řeší návrh a implementaci nástrojů, procesů a prostředků pro získaní správné a srozumitelné informace ve požadovaném čase. Tento proces zahrnuje všechny aktivity ohledně datového skladu - návrh, implementaci, aktualizace, správu apod.



Obrázek 8: Struktura datového skladu [12]

4.1.3.3 Datové tržiště (data mart) Datové tržiště je podmnožina datového skladu. Jedná se pohled na data specifická pro konkrétní obor/dotaz/proces. Každé datové tržiště musí být vlastním reprezentováno dimenzionálním modelem - tzn. obsahuje fakty i dimenze. Datová tržiště nemusí být nutně použita. Může existovat jediný celopodnikový datový sklad.

4.1.4 Analýza dat

Nyní máme data očištěna a uložena v datovém skladu - případně v datových tržištích. V této vrstvě chceme data změnit na informaci a rychle ji poskytnout danému uživateli. K tomuto byla vyvinuta speciální technologie OLAP (Online Analytical Processing), která umožňuje data uspořádat tak, aby je bylo možno srozumitelně předat uživateli. U primárních systémů se používá OLTP (Online transaction processing). Jedná se o způsob zpracování dat v transakčních systémech obvykle v relačních databázích. OLTP technologie je určena pro nasazení ve víceuživatelském systému s přihlédnutím k rychlosti a bezpečnosti. OLTP zajišťuje zpracování dat v reálném čase se zaměřením na jejich přidávání, úpravu a případně mazání. Typickým příkladem je například uložení objednávek internetového obchodu. Zatímco v OLTP ukládáme každou objednávku a položku zboží zvlášť, u datových skladů nás budou zajímat agregace - například součet cen prodaného výrobků v daném čase. Výstupem bývá zobrazen obvykle ve formě tabulky. Sloupce a

řádky jsou jednotlivé dimenze, hodnoty jsou pak fakty (measures). Například Microsoft Excel má možnost použít tzv. pivot table. To je tabulka určena přímo pro zobrazení dat z datové kostky, obsahující rozšířené možnosti jako řazení a filtrace - viz obrázek 9:

	A	B	C	D	E
1					
2					
3	ks na skladě	roky			
4	Výrobek	2008	2009	Grand Total	
5	Adaptér A2Z	0		0	
6	Adaptér A2Z 180	1		1	
7	Adaptér A2Z UNI	1		1	
8	Adaptér ASHIMA AU-31	0		0	
9	Adaptér AVID brzd. disc 203mm S/P zad.		2	2	
10	Adaptér BBB hlav.slož. externí		1	1	
11	Adaptér BOXXER kot. 203mm	0		0	
12	Adaptér CINELLI Quill	3		3	
13	Adaptér HOPE Y2K	1		1	
14	Adaptér MANITOU 185mm disk	0		0	
15	Adaptér MRP ISCG	1		1	
16	Adaptér PACE " V" RC36/38/39	2		2	
17	Adaptér PACE V-Brake RC31	1		1	
18	Adaptér RST SPACE pro rychloup.		2	2	
19	Adaptér SH brzd. disc P/S 203mm	2		2	
20	Adaptér SH levá část ang.záv.	0		0	
21	Adaptér SH pravá část ang.záv.	0		0	
22	Adaptér SIDI INSTEP	0		0	
23	Adaptér SIDI SPD-R	1		1	
24	Adaptér XSIV 200 mm	0		0	
25	Adaptér XSIV 203 mm	0		0	
26	Grand Total	13	5	18	
27					
28					
29					
30					
31					

Obrázek 9: Pivot table - Microsoft Excel

4.1.4.1 Rozdíly mezi OLAP a OLTP

OLTP:

- zpracovává aktuální data,
- vyžaduje se podpora transakcí,
- vysoký výkon pro operace INSERT/UPDATE/DELETE (vkládání, úprava, mazání dat),
- tabulky v normálové formě,
- provozní zpracování dat,
- detailní data,

- krátké dotazy náročné na rychlé provedení,
- desítky/stovky/tisíce záznamů,
- multiuživatelský přístup, tisíce uživatelů,
- velikost databáze max. jednotky GB.

OLAP :

- historická data,
- netřeba transakcí,
- operace INSERT/UPDATE/DELETE dávkově,
- bez normálové formy - redundance dat,
- analytické zpracování dat,
- agregace, sumy, ...,
- komplexní dotazy,
- statisíce/milióny záznamů,
- desítky uživatelů, není třeba řešit víceuživatelský přístup,
- velikost databáze v desítkách GB až TB.

4.1.4.2 Datová kostka [1, 4] OLAP se někdy používá jako synonymum k data warehousingu. Obvykle ale OLAP znamená nástroj, který pro uživatele získá data z datového skladu. OLAP zde zavádí datovou kostku. Jednotlivé nástroje definuji, spravují, ukládají kostku po svém, nicméně každý OLAP nástroj kostku využívá.

OLAP datová kostka se skládá z dimenzí (dimensions) a faktů (measures). Data v datovém skladě jsou z části agregovány, proto je musíme rozdělit na hodnoty, které chceme agregovat (fakty) - atributy, které chceme evidovat a které mají vliv na rozhodování ve firmě; obvykle to jsou numerické hodnoty a bývají agregované (součet prodaných kusů zboží, průměrná prodejní cena daného zboží v posledních pěti letech apod.).

Dále musíme určit hodnoty, podle kterých chceme agregovat (dimenze) - například čas prodeje výrobků (obvykle nechceme přesný čas jako v transakčních databázích, ale agregace například po měsících, letech, kvartálech) a kategorie výrobků. Výsledek pak je například, že výrobků z kategorie "plavky" se prodalo v červenci 1000 kusů, zatímco v listopadu 100 kusů. Zpracování výstupu (například křivka prodeje) je pak otázkou prezentační vrstvy BI. Dimenze obsahují atributy, podle kterých lze data agregovat. Produkt může chtít agregovat třeba podle barvy, kategorie, velikosti apod. Atribut datumu může být den, měsíc, kvartál, rok atd. Dimenze jsou spojeny s tabulkami faktů pomocí klíčů. Jedná se o tzv. surrogate klíč. Dimenze používají ještě jeden typ klíče a to tzv. natural.

Tento klíč je původní klíč z relační databáze - ID klienta, produktové číslo výrobku apod. V dimenzích obvykle existuje hierarchie - v některých případech jich může být víc. Některé hierarchie jsou vytvořeny automaticky: den-měsíc-rok - případně druhá podrobnější jako den-týden-měsíc-kvartál-půlrok-rok apod. Další lze vytvořit manuálně podle přání uživatele a typu dimenze. Například produkt - kategorie - podkategorie - produktová řada apod.

Někdy může být jeden atribut dimenze a někdy fakt. Příkladem může být věk, jako fakt by nás zajímal věkový průměr uživatelů. Jako dimenze pak například procentuální prodej výrobku vzhledem k věkovému rozmezí uživatelů.

Problémem OLAP přístupu je, že velikost dat (kostky) roste geometrickou řadou. Čím víc dimenzí a čím větší jejich granularita (čím víc atributů v dimenzi), tím je tento problém znatelnější.

4.1.4.3 Ukládání dat v OLAP. V OLAP technologii existuje několik způsobů ukládání dat. Z toho v praxi se nejčastěji používají dvě - případně jejich mírné modifikace.

- **MOLAP** (Multidimensional Online Analytical Processing) - ukládá hodnoty předpočítané agregované a to do multidimenzionální datové kostky. Jedná se o typický způsob ukládání dat vhodný pro malé až středně velké množství dat, nad kterými provádíme analýzy (dotazy) často. Aktualizace probíhají dávkově (dny, měsíce, roky atd.). Datová kostka vrací potřebná data velmi rychle - všechny kalkulace jsou již předpočítány. Datová kostka je optimalizována pro další operace nad ní - například: řez kostkou (dimenze aplikuje filtr na instance příslušné agregační úrovně dané dimenze). Výraznou výhodou může být, že s daty lze pracovat offline - bez připojení k primárním systémům. Všechna data jsou již předpočítána a uložena v kostce. Nevýhodou je náročnost na diskový prostor, nicméně tato nevýhoda v dnešní době nízkých cen pevných disků nebývá až tak podstatná. Poslední nevýhodou může být, že je třeba lidské zdroje na vytvoření takovéto kostky - oproti relačním databázím multidimenzionální databáze obvykle ve firmě nebývá a vytvoření datových pump pro různé formáty vstupních dat může být velmi náročné.
- **ROLAP** (Relational Online Analytical Processing) - jedná se o přístup, kdy není vyžadováno předpočítání dat. Místo toho OLAP přistupuje přímo do relační databáze a vytváří SQL dotazy, kterými agregace počítá real-time při každém požadavku uživatele. Protože počítání agregací v reálném čase je velmi náročné, je možné vytvořit dodatečné relační tabulky, kde lze některé vypočtené agregace uložit. Real-time výpočty agregací s sebou nesou riziko vysoké latence odpovědi - tzn. čekání uživatele na výsledek složitého výpočtu. Výhodou je real-time přístup k primárnímu systému a tudíž aktuálnost dat. Naopak zde vzniká problémem, že relační databáze jsou uzpůsobeny OLTP přístupu, tudíž počítání agregací může být velmi zdlouhavé a náročné na hardware. V tomto případě je třeba pečlivě hlídat výkonnost hardware, aby nedošlo (v krajním případě) k přetížení primární databáze a tím k problémům při běžných operacích jako vkládání nových dat z CMS/ERP či jiného IS. Prevence tohoto problému může být vytvoření kopie relačních tabulek na jiném

serveru. Oproti MOLAP, zde řez daty odpovídá v SQL dotazu podmínce WHERE. Tento způsob ukládání dat je vhodný pro velké objemy dat (například historická data), na které se moc nedotazuje.

- **HOLAP** (Hybrid Online Analytical Processing) - kombinace předchozích dvou přístupů. Část dat je uložena jako v případě MOLAP - v kostce a část dat je uložena v relačních databázích - ROLAP. HOLAP používá dva přístupy - vertikální, kdy jsou agregace uloženy v kostce pro zlepšení rychlosti při dotazování a detailní data jsou uložena v relačních databázích. Druhou možností je horizontální přístup - některé řezy (obvykle nejnovější dle času) jsou uloženy v kostkách. Zde se vychází z předpokladu, že na novější data se ptáme častěji a ostatní data jsou uložena pomocí ROLAP.

Pomocí Microsoft SQL Server Analysis Services, které jsem pro diplomou práci využil, lze nastavit způsob ukládání dat buď standardně na přednastavené kombinace přístupu a nebo čistě manuálně. Za zmínku stojí několik standardních způsobů:

- MOLAP - data jsou předpočítana a uložena v multidimenzionální databázi. Upozornění na aktualizaci dat v primárním systému je vypnuto, tudíž aktualizace je nutno provádět manuálně nebo dávkově v určitých intervalech.
- Scheduled MOLAP - stejně jako MOLAP, akorát aktualizace automaticky každých 24hodin
- Automatic MOLAP - zde je OLAP upozorněn na změnu dat v primárním systému a aktualizace dat je provedena automaticky
- Real-time HOLAP - fakty jsou uloženy v relačním formátu, agregace jsou uloženy v multidimenzionálním. OLAP je upozorněn na změnu dat v primárním systému. Všechny dotazy vrací aktuální výsledky.
- Real-time ROLAP - stejně jako HOLAP, nicméně všechna data jsou uložena v relační databázi.

Ručně je možno nastavit interval aktualize cache paměti, čas zahození cache, update cache, způsob uložení dat atd. Obdobně lze zvolit speciální přístup pro jednotlivé části kostky zvlášť. Například informace o stavu zboží na skladě mohou být real-time, celkové prodeje co 24 hodin apod.

4.1.4.4 Schémata datového skladu

Jak bylo uvedeno výše, ukládání dat v multidimenzionálních databázích se od relačních dat liší. Z tohoto důvodu se liší i schémata návrhu datového skladu. Nejpoužívanější jsou dva typy a to:

- Hvězdicové schéma (star) - se skládá z rozsáhlé centrální tabulky faktů a řadou malých doprovodných tabulek pro každou dimenzi. Jedna dimenze odpovídá právě jedné tabulce. Každá tabulka může mít více atributů - název, produktový kód,

barva... Jedná se o nejčastěji používané schéma převodu z relační do multidimenzionální databáze.

- Souhvězdí (Constellation) - některé aplikace mohou vyžadovat více tabulek faktů a ty potřebují sdílet tabulky dimenzí. Toto schéma může být zobrazeno jako soubor hvězd a proto se nazývá souhvězdí.
- Sněhová vločka (snowflake) - tabulky dimenzí jsou normalizovány. Díky tomu se data dělí do dalších tabulek a tím se sníží počet redundancí a nároky na diskový prostor. Taková tabulka je snadno udržovatelná. Tento typ schématu může zhoršit výkonost hledání dat, protože je třeba procházet více vnořených tabulek.

4.1.4.5 Metadata V podstatě se jedná o data, která popisují data. Metadata se nepoužívají jen v případě datových skladů. Jako příklad mimo tento obor se dá použít třeba fotografie v digitální podobě. Soubor kromě samotné fotky obsahuje i datum, typ fotoaparátu, někdy i GPS souřadnice místa focení atd. Metadata se vyskytují i v relačních databázích - například datový slovník, kde evidujeme datový typ, popis atributu atd. V datovém skladu je nutnost evidovat i data o jeho struktuře, obsahu, kdy byla provedena poslední aktualizace dat, rozdělení atributů do dimenzí atd. Metadata se dělí do několika skupin:

- Definování a mapování - popisuje význam jednotlivých atributů v datovém skladu z pohledu BI. Mapování popisuje odkud (tabulka, zdroj apod.) je daný atribut.
- Struktura dat - obsahuje datové typy dat, cizí/primární klíče a další. V relačních databázích by byl obdoba této skupiny metadat datový slovník databáze.
- Zdroje dat - obsahuje popis všech datových zdrojů kostky, včetně popisu jejich tabulek, atributů atd.
- ETL data - popisuje datovou pumpu, odkud čerpat data, kam zapsat, které úpravy aplikovat a jak často spouštět.
- Kvalita dat - popisuje pravidla kvality dat - které tabulky a atributy se kde používají, jaké akce spustit v případě nalezení chybových dat atd.
- Logování dat - popisuje výsledky všech procesů v datovém skladu - logování procesů, úpravy dat, zabezpečení, výsledky spuštění datové pumpy...
- Použití datového skladu - obsahuje logy využití datového skladu - kdo ke skladu přistupuje, na která data se dotazuje...

Metadata tedy používáme na popis dat pro koncové uživatele. Popisují, co ve skutečnosti znamená daný atribut dané dimenze. Tím se zvyšuje přehlednost datového skladu a datový sklad se líp udržuje. Dále metadata usnadňují práci vývojáři - popisují, kdy se děje která činnost a její výsledek. Například při chybě datové pumpy pomocí metadat zjistíme, které kroky byly použity a proč byl proces načítání dat neúspěšný.

4.1.4.6 MDX MDX (Multidimensional Expressions) je dotazovací jazyk pro OLAP databáze. Je podobný jako SQL pro dotazování dat v relačních databázích. MDX se zároveň používá na výpočty nad daty. Na rozdíl od SQL, MDX umožňuje tvořit multidimenzionální dotazy. Dále pak řezy kostkou, omezení výběru a v některých případech i změnu dat.

Základní klíčová slova:

- **SELECT** - zde slouží pro výběr atributů z dimenzí a v podstatě tím určujeme osy (v případě zobrazení do tabulky pak sloupce a řádky). Dále zde můžeme omezit výběr dat (určité roky, jména uživatelů apod.)
- **FROM** - určuje kostku, na kterou se dotazujeme.
- **WHERE** zde na rozdíl od SQL nefunguje jako omezení (podmínka) výběru dat, ale jako řez kostkou. Omezení výběru dat v případě MDX najdeme v části SELECT

Základní syntaxe:

```
SELECT < dimenze/fakty > ON <osa1>, <dimenze/fakty> ON <osa2> FROM <kostka>
```

Výpis 22: Základní syntaxe

Například tento dotaz (23) vrátí průměrnou nákupní cenu zboží ve skladech v závislosti na čase. NON EMPTY znamená, že se vrací pouze neprázdné řádky. Zde nula není brána jako prázdná hodnota. Prázdné je zde myšleno ve smyslu bez dat - například když není ani jeden záznam pro určitý měsíc, tento řádek se vůbec nevypíše. Pod dotazem najdeme možnost zobrazení výstupu dotazu - obrázek 10.

```
SELECT
NON EMPTY {[Sklad].[Nazev].ALLMEMBERS} ON COLUMNS,
NON EMPTY {[Cas].[Mesic – nazev].ALLMEMBERS} ON ROWS
  FROM [Datova kostka]
WHERE ([Measures].[Nakupni Cena Prumerna])
```

Výpis 23: Ukázkový dotaz - dvě dimenze

	Břeclav - Hlavní	Hlavní	Ostrava - Hlavní sklad	Ostrava - Komisní sklad	Ostrava - Maloobchod
December 2008	0	0	0	0	0
February 2009	0	0	0	0	0
January 2009	115	115	115	115	115
March 2009	25	25	25	25	25
November 2008	0	0	0	0	0
October 2008	250	250	250	250	250
October 2009	1500	1500	1500	1500	1500
September 2008	0	0	0	0	0

Obrázek 10: Výstup dotazu ve Visual Studiu

MDX nám umožňuje i filtrování dat. Chceme-li například zjistit průměrnou nákupní cenu zboží jen na skladech v Ostravě (= obsahující v názvu skladu slovo ”Ostrava”):

```
SELECT
NON EMPTY
(
    Filter ([Umisteni Karty].[Nazev].[Nazev].ALLMEMBERS,
    Instr ([Umisteni Karty].[Nazev].currentmember.Properties('Member_Caption' ),'Ostrava) = 1)
) ON COLUMNS,
NON EMPTY {[Cas].[Mesic - nazev].ALLMEMBERS}  ON ROWS
  FROM [Datova kostka]
WHERE ([Measures].[Nakupni Cena Prumerna])
```

Výpis 24: Ukázkový dotaz - filtrování dimenze

Další možnosti by bylo například konkrétní omezení času, případně pro každý sklad/měsíc vidět průměrnou nákupní cenu pro každý konkrétní výrobek zvlášť, vypočítat kolik procent z celkové průměrné nákupní ceny je na daném skladě apod. Vše toto MDX zvládá. Nicméně k takovým operacím je třeba zavést oproti SQL nové datové typy. Příklad datových typů v MDX dotazech:

- Skalární datový typ (Scalar) - může být číslo nebo řetězec (string).
- Dimenze - zde máme přístup ke všem atributům, případně konkrétním hodnotám. MDX nerozeznává vazby mezi dimenzemi automaticky. Značíme do hranatých závorek [] - [Cas], [Produkt] apod.
- Stupeň zanoření (level) - stupeň zanoření v určité dimenzi - *[Cas].[Rok].[Kvartal].[Mesic]* apod.
- Člen (Member) - jedná se přímo o konkretní hodnotu atributu dimenze. Například *[Cas].[Mesic - nazev].&[prosinec]*. Můžeme zde používat funkce jako PrevMember (předchozí), FirstChild (první) apod.
- Set - skupina členů z jedné nebo více dimenzí. Tato skupina se píše do složených závorek . Chceme-li získat konkrétní hodnotu atributu, použijeme anglický znak pro AND - "&". Příkladem může být požadavek na konkrétní produkty s produktovým číslem 1002 a 1003: *[Produkt].[Cislo].&[1002], [Produkt].[Cislo].&[1003]* Nechceme-li vypisovat výpis všech hodnot, můžeme použít funkci "ALLMEMBERS". Tento dotaz vrátí všechny názvy měsíců: *[Cas].[Mesic - nazev].ALLMEMBERS*
- Řez (Tuple) - část (výřez) kostky - píše se do kulatých závorek (). Například chceme-li vybrat z produktů všechna kola: *[Produkt].[Vsechny produkty].[Kolo]*
- Vypočítané hodnoty (Calculated members) - potřebujeme-li nad určitou skupinou dat vypočítat například rozdí, součet, procentuální podíl, zaokrouhlení nebo například rozdělení hodnot do skupin, použijeme práve tento datový typ. Zde se používají klíčová slova *WITH MEMBER [název] AS [výpočet]*. Například rozdíl mezi vydanými a přijatými fakturami v korunách: *WITH MEMBER [Faktura].[Rozdil] AS '[Measures].[Cena Celkem - Faktura Vydana] - [Measures].[Cena Celkem - Faktura Prijata]', SELECT [Faktura].[Rozdil] ON COLUMNS.*. Příklad na výpočet procent

- kolik procent uživatelů systému je mužského/ženského pohlaví: *WITH MEMBER [Osoba].[Pohlavi procent] AS '([Measures].[Osoba Count]) / ([Measures].[Osoba Count],[Osoba].[Pohlavi].[All]) * 100'*

4.1.5 Prezentace dat

Tato poslední vrstva BI se zabývá zobrazením již nalezených informací uživateli. Existuje nepřeberné množství způsobů zobrazení, nejčastěji dle mého však převládá tabulka a z ní případně graf. Stejný způsob jsem použil i v mé diplomové práci. Přístupů pro vytvoření takového výstupu je několik. Například tabulky a grafy v programech (účetní programy) nebo přímo pomocí tabulkového procesoru - například pivot-tables v MS Excel (viz výše), který je součástí MS Office. webové/desktopové reporty. Microsoft pro reporty má velmi mocný a rozsáhlý nástroj - SQL Server Reporting Services (SSRS). Jedná se o server-side software pro generování reportů z různých datových zdrojů. Samozřejmostí je propojení s datovou kostkou v Microsoft SQL Server Analysis Services. Reporty lze tvořit přímo ve Visual Studiu. To obsahuje průvodce pro připojení ke zdroji dat, vytvoření dotazu (možnost přímo vytvoření MDX dotazu) přes Designer až po zobrazení do tabulky, několika druhů grafu apod. Reporty lze na server uložit přímo, například ve spojení s Sharepoint se jedná o mocný nástroj pro firmy a jejich manažery. Popis SSRS by vydal na samotnou práci, proto zde již nebudu popisovat další podrobnosti.

Jednou z další možností je se připojovat se na datovou kostku a dotazovat se ji přímo podobně jako na SQL server. Tento způsob jsem využil ve své práci a to hlavně z důvodů dynamického generování MDX dotazů - uživatel si vybírá a filtruje data, která chce zobrazit. Výsledkem je MDX dotaz, který se zobrazí do tabulky a dále volitelně do grafu. Na dotazování kostky používám technologii ADOMD.NET.

4.1.5.1 ADOMD.NET [13] Jedná se o rozšíření ADO.NET, která se používá v .NET pro dotazování na MS SQL databázi. V dnešní době bývá nahrazena LINQ-em, nicméně stále se nejedná o zastaralou technologii. ADOMD.NET je prakticky jediným způsobem, jak v kódu získat data z OLAP databáze. Jedná se o framework data provider (poskytovatel), který komunikuje s MS Analysys services pomocí TCP/IP nebo HTTP protokolu. Jako jazyk pro dotazování můžeme použít MDX pro dotazování kostky, případně DMX (Data Mining Extensions) pro dolování dat.

Důležité objekty ADOMD.NET:

- AdomdConnection - objekt řešící připojení k datové kostce. Připojení se automaticky neuzavírá, je třeba volat metodu Close() nebo Dispose() ručně v kódu. Obsahuje nastavení connection stringu (textový řetězec definující připojení k OLAP databázi)
- AdomdCommand - reprezentuje dotaz, který bude odeslán na server. Objekt může být použit pro vrácení výsledku pro AdomdDataReader
- AdomdDataReader - pouze ke čtení a pouze po záznamech postupně (forward-only - nelze se vracet zpátky nebo v záznamech přeskakovat). Tento objekt může zvýšit

výkonnost aplikace, protože výsledek načítá hned jak je k dispozici místo čekání na celkové vyhodnocení dotazu.

- AdomdDataAdapter - pouze ke čtení z datového zdroje - například načtení výsledku z AdomdCommand a vložení dat do DataSetu.

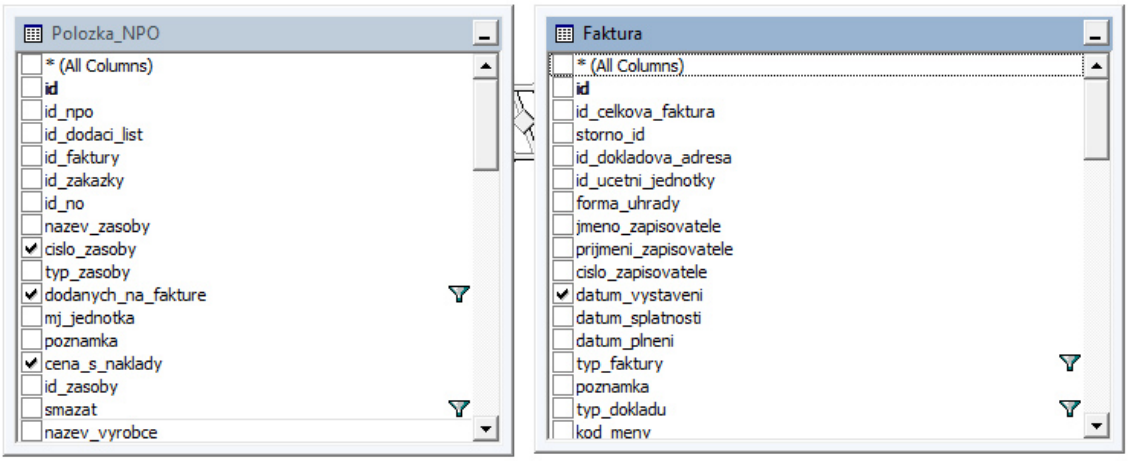
5 Implementace

Implementace modulu byla rozdělena do dvou částí. První částí bylo samotné vytvoření datové kostky - dimenze, fakty a to podle potřeb firmy. Jednalo se především o požadavky ekonomického směru přímo od klienta ERP systému. Mezi základní požadavky patřily například součty DPH, nákupy, příjmy, zboží na skladech, obrátkovost (prodejnost) zboží apod., jak je zmíněno v kapitole 2. Z původní primární databáze bylo třeba pomocí pohledů (views) získat potřebná data. Pohledů bylo vytvořeno několik, zde si vysvětlíme jeden komplexní, navíc složený ze dvou "pod-pohledů". Obdobně u samotné relační databáze. Původní databáze ERP obsahovala přes 100 tabulek. Proto není vhodné všechno popisovat a případně zobrazit pomocí E-R diagramu. Vybereme zde jen určité úseky na konkretním příkladu. Celé schéma je možno vidět přímo v databázi na SQL serveru. Schéma datové kostky pak při otevření datové kostky ve Visual Studiu.

5.1 Příklad vytvoření faktu

Například úkolem bylo zjistit obrátkovost (prodejnost) jednotlivých druhů zboží v procentech v závisloti na čase. Výsledek měl být například, že daný výrobek se prodal v 5% případů v daném čase (den, měsíc, rok, atd.). Pro datovou kostku bylo potřeba vytvořit ve zdrojové databázi pohled, který by obsahoval počet prodaných kusů pro každý výrobek a k tomu cenu za kus. Tento pohled dále musel obsahovat cizí klíče pro propojení s ostatními dimenzemi. Konkrétně *cislo_karty* pro propojení s dimenzí *Karty*. Počet kusů a jejich cena se nachází v tabulce *Polozka_NPO*. Názvy atributy *dodanych_na_fakture* a *cena_s_naklady* jsou výmluvné samy za sebe. Ve stejné tabulce se nachází i atribut *cislo_zasoby*, který by se měl správně jmenovat *cislo_karty*. Vše lze přehledně vidět na obrázku číslo **??**.

Zde byla chyba už v návrhu pojmenování atributů v původní databázi. Atribut *cislo_zasoby* tedy v tomto případě značí cizí klíč pro spojení tabulky *Polozka_NPO* s tabulkou *Karta*. Čili v této fázi bylo v pohledu kolik kusů které karty bylo prodáno za jakou cenu. Pro potřeby zadání chyběla už pouze závislost na čase. Datum prodeje obsahovala tabulka *Faktura*. Zde bylo třeba krom spojení *Karta - Faktura* vyřešit situaci, kdy tabulka *Faktura* obsahuje přijaté i vydané faktury a ty dále dělí na dopropisy a běžné faktury. Naštestí tato situace znamenala pouze rozšíření pohledu i podmínku WHERE a o podmínky na atributech *typ_dokladu* a *typ_faktury*. Zároveň bylo vyfiltrovány chybné položky. Ty položky, které měly prodaných kusů nula, nebo neměly číslo faktury. Je zřejmé, že chyby vznikly při vývoji ERP systému. I když chyby ve fakturách byly opraveny, databáze obsahovala dále chybná data z doby před opravou. Pro přehlednost diagram tabulek a dotaz pohledu (zde pro přehlednost neuvádím celou syntaxi pohledu *CREATE VIEW...* pouze část výberu dat). Je třeba uvést, že byly vybrány pouze nesmazané položky. ERP položky nemaže přímo (příkaz DELETE) pouze je označí jako smazané pomocí atributu *smazat = 1*. Tyto položky je tedy třeba vyfiltrovat.



Obrázek 11: Tabulky obsahující data pro pohled

```
SELECT   TOP (100) PERCENT dbo.Polozka_NPO.dodanych_na_fakture AS dodanych, dbo.
    Polozka_NPO.cena_s_naklady AS cena, dbo.Polozka_NPO.cislo_zasoby,
                    dbo.Faktura.datum_vystaveni
FROM        dbo.Polozka_NPO INNER JOIN
                    dbo.Faktura ON dbo.Polozka_NPO.id_faktury = dbo.Faktura.id
WHERE   (dbo.Faktura.smazat <> 1) AND (dbo.Faktura.typ_dokladu = 1) AND (dbo.Faktura.
    cislo_dokladu IS NOT NULL) AND (dbo.Polozka_NPO.dodanych_na_fakture > 0) AND
                    (dbo.Faktura.typ_faktury = 0) AND (dbo.Polozka_NPO.smazat = 0)
```

Výpis 25: Výběr dat pro pohled

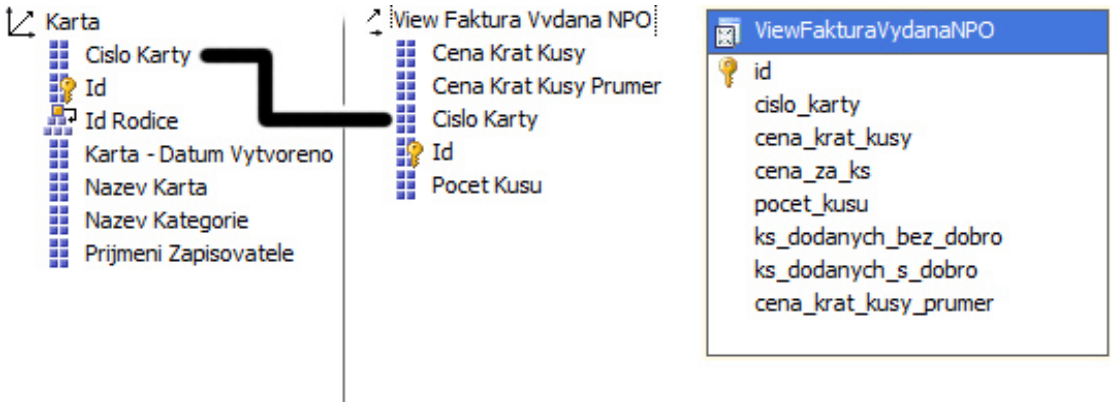
Vytvořený pohled z ukázky číslo 25 je pouze první částí. Aktuální pohled totiž nezohledňoval dobropisy. Čili od cen a kusů z nově vytvořeného pohledu bylo třeba odečíst hodnoty z dobropisů. Dobropis se rozeznával pomocí atributu *typ_faktury*. Dotaz pro pohled dobropisů byl jinak stejný s předchozím. Ve výsledný pohled po odečtení dobropisu. Pro usnadnění výpočtu byl vynásoben počet prodaných kusů s cenou zboží přímo v pohledu a přidán jako další atribut (sloupec) pohledu. Dále byla dopočítána průměrná cena za kusy pro použití v kostce. Fuknce *ISNULL* vrací při neexstující (null) hodnotě nulu. Toto ošetření bylo nutné proto, aby pohled nezahazoval data. Pokud by se odečítala hodnota null od konktrétní hodnoty faktury, výsledek by byl null, čili žádný řádek (kostka hodnoty null ignoruje). Zatímco díky této funkci se vracela hodnota nula, která po odečtení od ceny faktury původní cenu zachovala. Zde je třeba si uvědomit, že nula je brána jako matematická nula, zatímco null v MS SQL znamená prázdný atribut (= bez číselné hodnoty). Výsledný pohled 26:

```
SELECT   TOP (100) PERCENT v1.cislo_zasoby AS cislo_karty, SUM(ISNULL(v1.dodanych, 0) *
    ISNULL(v1.cena, 0) - ISNULL(v2.dodanych, 0) * ISNULL(v2.cena, 0))
                    AS cena_krat_kusy, AVG(ISNULL(v1.dodanych, 0) * ISNULL(v1.cena, 0) -
                        ISNULL(v2.dodanych, 0) * ISNULL(v2.cena, 0)) AS
                        cena_krat_kusy_prumer,
```

```
                SUM(ISNULL(v1.cena, 0) − ISNULL(v2.cena, 0)) AS cena_za_ks, SUM(
                    ISNULL(v1.dodanych, 0) − ISNULL(v2.dodanych, 0)) AS pocet_kusu,
                    SUM(ISNULL(v1.dodanych, 0))
                AS ks_dodanych_bez_dobro, SUM(ISNULL(v2.dodanych, 0)) AS
                    ks_dodanych_s_dobro, karta.id, v2.datum_vystaveni
FROM        dbo.ViewFakturaVydanaNPO_BEZdobropis AS v1 LEFT OUTER JOIN
                dbo.ViewFakturaVydanaNPO_dobropis AS v2 ON v1.cislo_zasoby = v2.
                    cislo_zasoby LEFT OUTER JOIN
                dbo.ViewKarta AS karta ON karta.cislo_karty = v1.cislo_zasoby
WHERE   (v1.cislo_zasoby IS NOT NULL) AND (karta.smazat <> 1) AND (v2.datum_vystaveni =
    v1.datum_vystaveni)
GROUP BY v1.cislo_zasoby, karta.id, v2.datum_vystaveni
```

Výpis 26: Výsledný pohled

Nyní stačilo přidat atributy *cena_krat_kusy*, *cena_krat_kusy_prumer* jako fakty (measures). Dále byl přidán pohled jako dimenze a pomocí cizího klíče spojen s již vytvořenou dimenzí karty. Na obrázku 12 je vidět dimenze karty, naznačeno spojení s nově vytvořenou dimenzí. Na obrázek byla přidána i původní zdrojová relační tabulka.



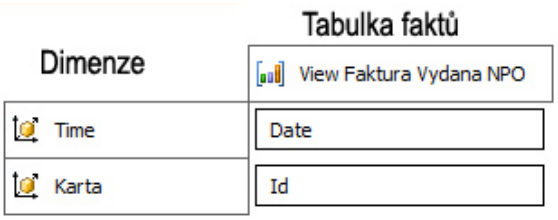
Obrázek 12: Dimenze a zdrojová relační tabulka

Spojení dimenzí se ve Visual Studiu nastavuje v záložce *Dimension Usage*. Vždy se nastavuje spojení dimenze s konkrétní fakty. Existuje několik druhů spojení dimenzí:

- **No relationship** - žádné spojení.
- **Regular** - spojení přímo pomocí cizího klíče - atributů.
- **Fact** - dimenze je zároveň tabulka faktů.
- **Referenced** - dimenze je spojena s tabulkou faktů přes jinou/jiné dimenze.Například faktura obsahuje cízí klíc na tabulku adresa. Adresa je spojena s tabulkou zemí. Tabulku země spojimí přes adresu s fakturou a nyní můžeme faktury rozdělit podle země vydání.

- **Many-to-many** - dimenze je spojena vazbou N:N. Například výrokeb patřící do více kategorií.
- **Data mining** - dimenze je vytvořena pomocí dolování dat.

Pro spojení dimenze karta a nové dimenze bylo použito spojení *Regular* a atributy *cislo_zasoby* a *cislo_karty* a obdobně byla spojena fakty s tabulkou času. Po spojení je možno fakty rozdělit pomocí zmíněných dimenzí a všech jejich atributů. Spojení ve Visual Studiu je možno vidět na obrázku 13.



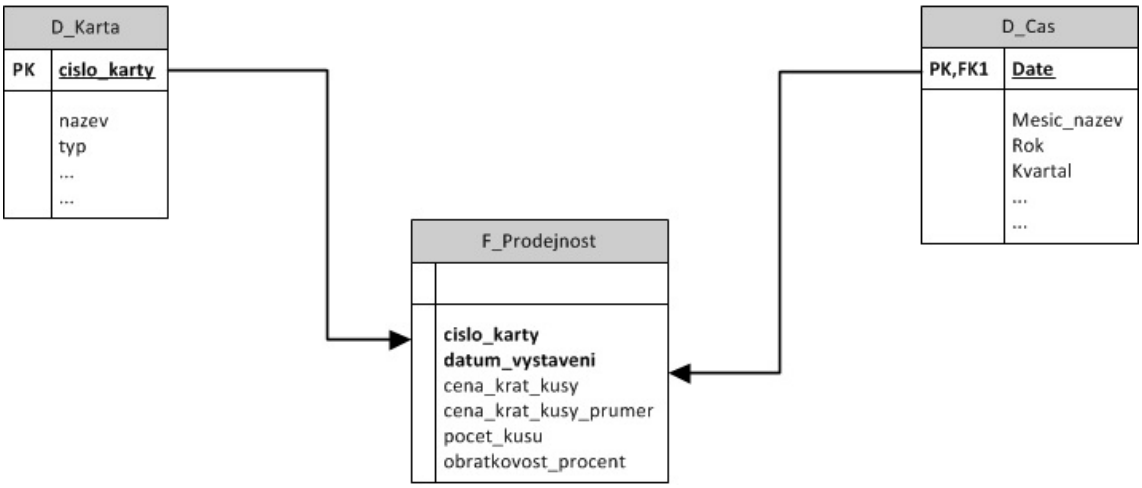
Obrázek 13: Spojení tabulky faktů s dimenzemi

Vetšina hodnot v kostce byla vyjádřena v čáskách. U některých bylo třeba vypočítat procentuální zastoupení (které zboží se prodává nejvíc - například kolo typu X se prodalo v 10% případů apod.). Na tyto výpočty byly použity calculated members. Calculated members lze vytvářet pomocí MDX dotazů - viz ukázka kódu číslo 27. Pro usnadnění vývoje kostky Visual studio nabízí jednoduché grafické rozhraní pro jejich tvorbu. Nicméně samotný výpočet je třeba zapsat ručně přímo pomocí MDX. Rozhraní pouze usnadňuje zařazení nového atributu (measure) do skupiny, jejich přehledný výpis apod. Procenta vypočteme pomocí jednoduchého MDX dotazu, který lze vidět dále. Celkovou částka byla vyděléna počtem položek. Jako položky (a tím i agregovat hodnoty například podle měsiců) je možno použít číslo karty, název karty a/nebo měsíce/roky. Výsledek je třeba poté vynásobit stem, aby byl výsledek správně v procentech, viz výpis 27.

```
([Measures].[Cena Krat Kusy]) /
(
[Measures].[Cena Krat Kusy],
[Karta].[Cislo Karty].[All],
[Karta].[Nazev Karta].[All],
[Cas].[Mesic - nazev].[All]
[Cas].[Rok - nazev].[All]
) * 100
```

Výpis 27: Calculated member - procenta z prodeje

Na obrázku 14 lze vidět schéma části kostky. To obsahuje dvě dimenze - Karta a Čas. Dále pak jednu tabulku faktů - Prodejnost. Jedná se o typ hvězda (popsáno výše). Obdobně byly vytvořeny ostatní fakty a dimenze. Vždy dle konkrétního požadavku firmy.



Obrázek 14: Schéma části kostky

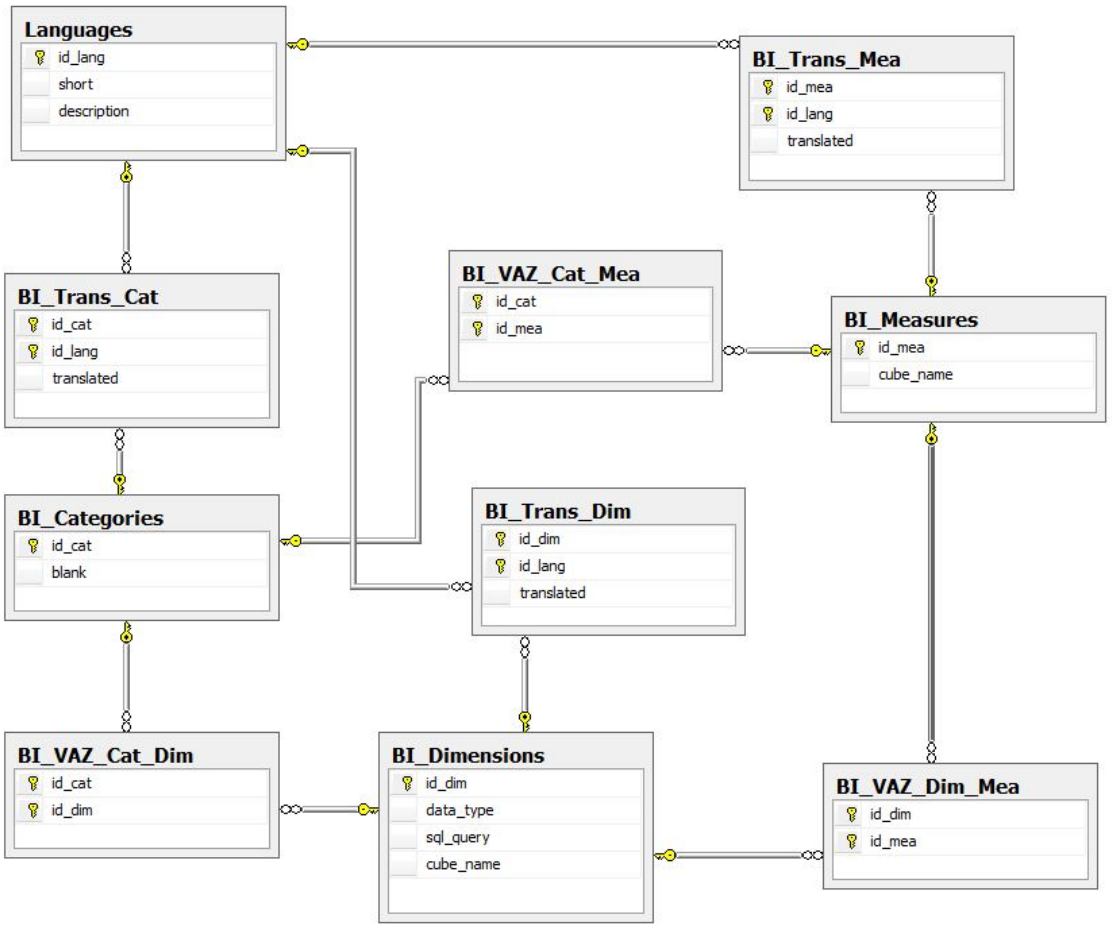
5.2 Webové rozhraní

Druhou částí úkolu bylo vytvořit webové rozhraní pro získaná a zobrazení dat z kostky. Vzhledem k tomu, že úkolem bylo, aby si klient mohl sestavovat dotazy na kostku v podstatě libovolně, nebylo vhodné použití Microsoft SQL Server Reporting Services. Ty jsou vhodnější pro tvoření statických reportů. Rozhraní je rozděleno na dvě části. První částí je výběr hodnoty (faktů) a filtrů (dimenzí) - v podstatě se jedná o sestavení MDX dotazu uživatelem pomocí jednoduchého webového rozhraní. Tyto data se načítájí z vlastní databáze - díky tomuto řešení je možno pro každého klienta nastavit fakty a dimenze podle přání. Samozřejmostí je jejich zařazení do kategorií. Dimenze lze dále základními způsoby filtrovat. Pro číselné hodnoty větší, menší, rovno apod. Pro řetězce pak filtry typu obsahuje, neobsahuje apod. Filtry lze kdykoliv rozšířit. V druhé části pak probíhá samotné zobrazení do tabulky a volitelně grafu.

5.2.1 Tvorba dotazu na datovou kostku

Jak je uvedeno výše, výběr faktů a dimenzí se provadí z vlastní databáze. Výběr dimenzí je pak omezen podle vybraného faktu. Například zvolí-li uživatel zobrazení počtu zboží na skladě, nezajímá ho dimenze faktura a dělení podle faktur. Toto omezení tedy zvyšuje přehlednost aplikace pro uživatele. Dimenze může samozřejmě záviset na více faktech (vztah 1:N). Zároveň je webové rozhraní připraveno na multijazyčnost původního systému. Dimenze i fakty mohou být přeloženy do více jazyků.

Na obrázku 15 vidíme schéma databáze webového rozhraní. Tabulka *Languages* obsahuje seznam jazyků v původním systému. Tato tabulka bude společná pro rozhraní i ERP. Tabulky obsahující *"_Trans_"* jsou vazební tabulky pro překlad. Tabulky obsahující *"_Vaz_"* jsou vazební tabulky, které obsahují závilosti na kategoriích, respektive dimenzích na faktech. Tabulka *BI_Measures* obsahuje seznam faktů. Atribut *cube_name* obsahuje název faktu



Obrázek 15: Schéma databáze webového rozhraní

přímo v kostce - tento atribut je tedy použit při tvorbě MDX dotazu. Tabulka *Bi_Categories* obsahuje kategorie. Ty byly vymyšleny zadavatelem a složí čistě pro usnadnění orientace uživatelů v rozhraní. Kategorie se do kostky nijak nepromítají. Kategorie může obsahovat libovolně faktů/dimenzí. Tabulka *BI_Dimensions* obsahuje seznam dimenzí. Stějne jako u faktů tabulka obsahuje atribut *cube_name*, který obsahuje název dimenze v kostce. Atribut *data_type* pak obsahuje datový typ dimenze. Ten se využívá pro uživatele při nabídce filtrů při výběru dimenzí. Na práci s databázi byla použita technologie LINQ - teoreticky popsáno v předchozí části. Firma má zakoupené komponenty od firmy Telerik a podmínkou bylo je využít všude, kde byla možnost. Tato podmínka byla požadována hlavně z důvodu sjednocení kompoment v ERP a webového rozhraní. Navíc tyto komponenty přináší víc možností a jsou přívětivější pro uživatele.

Uživatel v prvním kroku vybere z dostupných kategorií pro fakty. Ty se načítají při načtení stránky v metode *Page_Load()*. Jelikož načítání hodnot do komponent je z vetší

části stejné, uvedu a popíši zde celý postup. V dalších krocích si popíšeme už jen čistě LINQ dotaz na databázi.

```
// razeni v CB
int i = 1;
// pridani vychozi polozky do CB
CB_mea_cat.Items.Insert(0, new RadComboBoxItem("", "0"));

// nacteme kategorie
var categories =
    from a in db.BI_Categories
    from b in db.Languages
    from c in db.BI_Trans_Cats
    from d in db.BI_Measures
    from f in db.BI_VAZ_Cat_Meas
    where a.id_cat == c.id_cat && b.id_lang == c.id_lang && b.@short == user_lang
      && d.id_mea == f.id_mea && f.id_cat == a.id_cat
    select new { a.id_cat, c.translated };
// kazdou kategorii chceme jen jednou
categories = categories.Distinct();
// pridani polozek z db (poradi, (text, value) )
foreach (var x in categories)
{
    CB_mea_cat.Items.Insert(i++, new RadComboBoxItem(x.translated, x.id_cat.ToString()));
}
// a mame je tam...
CB_mea_cat.DataBind();
```

Výpis 28: Načtení kategorií faktů

Jak je vidět v ukázce 28, pomocná proměnná *i* zde obstarává řazení hodnot v komponentě. Jako první hodnota byl přidán prázdný řádek. Díky tomu uživatel vidí, kde už má hodnoty vybrány a kde ne. Prázdný řádek je zde výchozí hodnotou po načtení a zároveň v kódu je dle něj rozpoznán stav, kdy uživatel žádnou hodnotu nevybral. Metoda *Insert* u ComboBox komponenty obsahuje dva atributy. Prvním je zmíněné pořadí. Druhým je nová položka komponenty, která obsahuje taktéž dvě proměnné. První je textový řetězec, který bude zobrazený uživateli. Druhou je pak hodnota, která bude poslána na pozadí a určuje výběr uživatele (v HTML formulářích to odpovídá atributu *value*). Jako příklad evedeme, že kategorie se může jmenovat *Faktury* a na pozadí se pošle její id v databázi. Přesně tak funguje přidávání kategorií na předposledním řádku. *...new RadComboBoxItem(x.translated, x.id_cat.ToString());*. První proměnná je přeložený název kategorie získaný z databáze. Druhům je pak id kategorie. Jelikož hodnota může být i řetezec, očekává se, že proměnná bude datového typu string (řetězec). Toho se dosáhlo pomocí *ToString()*, což je metoda určena pro tento případ přímo v .NET.

Načtení kategorií z databáze bylo provedeno jedním LINQ dotazem. Na první pohled se jedná o složitější dotaz, protože bylo třeba projít více tabulek a to hlavně díky překladu. Dotaz měl vracet všechny kategorie, které obsahují alespoň jeden fakt. Jinými slovy bylo vynechány prázdné kategorie. Konktrétně bylo potřeba získat ID kategorie a překlad dané kategorie. Zapsáno databázově (formát Tabulka.atribut budu používat i nadále -

tento zápis se používá také přímo v SQL) bylo potřeba získat atributy *BI_Categories.id_cat* jako ID kategorie a *BI_Trans_Cats.translated* jako překlad do daného jazyka.

Popis podmínky WHERE se dá rozdělit na tři části. V první části (pro přehlednost proměnné přepsány na názvy tabulek) *BI_Measures.id_mea == BI_VAZ_Cat_Meas.id_mea && BI_VAZ_Cat_Meas.id_cat == BI_Categories.id_cat* byly získány všechny neprázdné kategorie. Byly vybrány všechny fakty (measures) a spojeny přes vazební tabulku *BI_VAZ_Cat_Meas* s tabulkou kategorií. Nyní bylo třeba tyto neprázdné kategorie přeložit. Z tabulky jazyků byl vybrán aktuální jazyk uživatele, načítá se na začátku skriptu do proměnné *user_lang*. Protože atribut obsahující zkratku jayku v databázi je zároveň i klíčové slovo pro LINQ, bylo třeba jej opatřit znakem zavináč "@". Tím bylo řečeno, že se jedná o atribut databáze. Celá část dotazu na omezení jazyku je pak jedna podmínka *b.@short == user_lang*. Posledním krokem bylo pro nalezené (neprázdné) kategorie zjistit překlad. Jinými slovy bylo třeba přes vazební tabulku *BI_Trans_Cats* mezi kategoriemi a jazykem najít překlad. *BI_Categories.id_cat == BI_Trans_Cats.id_cat && Languages.id_lang == BI_Trans_Cats.id_lang* - vybrat všechny překlady, které obpovídají ID kategorie.

Tento dotaz může vracet jeden název kategorie víckrát. Proto je třeba po jeho provedení odeprat duplicitní kategorie. Na toto existuje v LINQ metoda *Distinct()*, která vrátí každou kategorii pouze jednou. Dále již stačilo pouze výsledek dotazu - projít všechny řádky a kategorie přidat do komponenty. Posledním krokem je pak zavolaní metody *DataBind()*, která nastaví přidané hodnoty do komponenty.

Po vybrání kategorie se uživateli otevře možnost vybrat fakt, který chce zobrazit. Načtení možných faktů probíhá obdobně.

```
        // nacteni ID kategorie
        int id_cat = Int32.Parse(CB_mea_cat.SelectedValue);
        // dotaz na data
        var measures =
    from a in db.BI_Categories
    from b in db.Languages
    from c in db.BI_Trans_Meas
    from d in db.BI_Measures
    from f in db.BI_VAZ_Cat_Meas
    where d.id_mea == c.id_mea && b.id_lang == c.id_lang && b.@short == user_lang
      && d.id_mea == f.id_mea && f.id_cat == id_cat
    select new { c.id_mea, c.translated };
```

Výpis 29: Načtení faktů podle zvolené kategorie

Jak můžeme vidět v ukázce číslo 29, proměnná *id_cat* označuje ID kategorie zvolené v předchozím kroku. Byla načtena pomocí *int id_cat = Int32.Parse(CB_mea_cat.SelectedValue);*. *CB_mea_cat* je název komponenty pro výběr kategorie. Vlastnost *SelectedValue* vrací hodnoty vybranou v této komponentě (právě to ID kategorie). *Int32.Parse()* je pak převedení hodnoty do int (celé číslo). Dále byla vytvořena v dotazu podmínka, která načetla ID všech measures (faktů) v dané kategorii - *d.id_mea == f.id_mea && f.id_cat == id_cat*. Zjištění překladu bylo rozepisáno výše, jedná se o stejný princip jako u kategorií i v případě faktů. Vybral-li uživatel fakt, získává možnost vybrat si dimenze, kterými chce fakt agregovat.

Výběr dimenzí byl naprogramován pomocí user control. Jedna user control řeší jednu dimenzi a načítání jejich kategorií, filtrů apod. Na stránku lze přidat více user control - čili

uživatel může přidávat více dimenzí. Výhodou tohoto řešení je, že se nemusí generovat několik skupin komponent, ale vždy pouze jedna user control pro jednu danou dimenzi (řádek tabulky). User control má příponu *.ascx a jedná se v podstatě o programátorem vytvořenou komponentu. User control může obsahovat jiné komponenty, proto je její vytvoření relativně jednoduché. Tvoří se podobně jako nová webová stránka - má vlastní zobrazení (interface) a kód na pozadí. Při vývoji můžeme použít například i designer pro náhled a další nástroje Visual studia. V user control můžeme komponenty ovládat přímo z kódu user control a zároveň k nim máme přístup i z hlavní stránky. User control však nemůže běžet sama o sobě, musí být umístěna na nějaké stránce.

Výběr dimenzí je závislý na výběru faktů (ukázka 30). Načítání dimenzí a jejich kategorií je obdobné jako v případě faktů. Dle zvoleného faktu se načtou všechny kategorie, ktere nejsou prázdné a zároveň obsahují povolené dimenze. Takový dotaz by se v SQL skládal z poddotazu - *SELECT ... FROM ... WHERE* ***atribut IN (SELECT....)*** .

```
SELECT Distinct(BI_Trans_Cat.translated, BI_Trans_Cat.id_cat ) FROM BI_Trans_Cat,
    BI_Categories, BI_VAZ_Cat_Dim, Languages
        WHERE BI_Categories.id_cat = BI_Trans_Cat.id_cat
          AND BI_VAZ_Cat_Dim.id_cat = BI_Categories.id_cat
          AND Languages.short = 'cz'
          AND Languages.id_lang = BI_Trans_Cat.id_lang
          AND BI_VAZ_Cat_Dim.id_dim IN (SELECT id_dim FROM BI_VAZ_Dim_Mea
                                                WHERE id_mea = 1)
```

Výpis 30: Zjištění kategorií obsahující dimenze v závislosti na faktu - SQL

V poddotazu *...BI_VAZ_Cat_Dim.id_dim IN (SELECT id_dim FROM BI_VAZ_Dim_Mea WHERE id_mea = 1)* byly vybrány všechny ID dimenzí, které se vztahují (jsou povolené) vzhledem k uživatelem vybranému faktu. Zbývající část dotazu je opět obdobná jako výše. Pro všechny vybrané dimenze byly zjištěny kategorie a ty přeloženy do zvoleného jazyka (v příkladu do čestiny). Opět bylo třeba odebrat duplicitní položky. K tomu byla použita SQL funkce *Distinct()*.

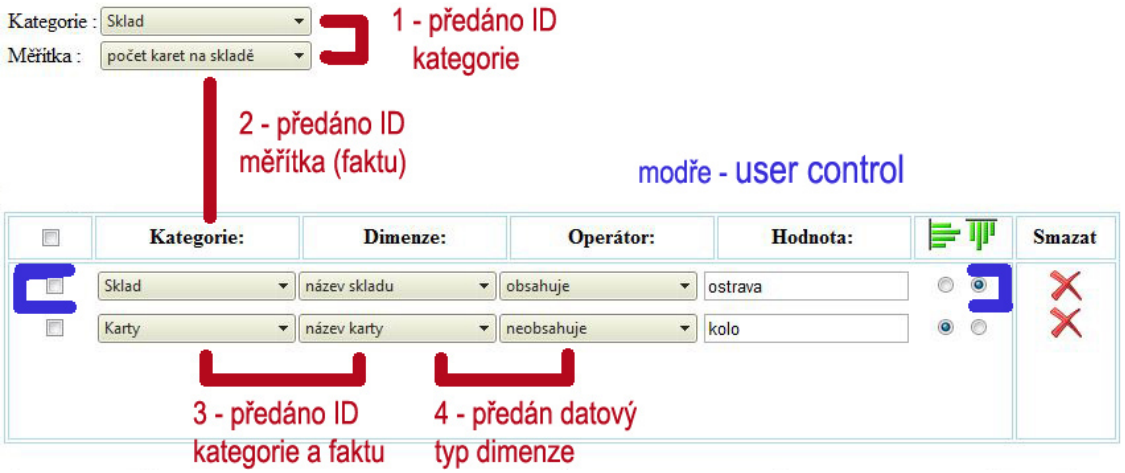
Bohužel LINQ neumí vytvářet poddotazy. Zde bylo třeba rozdělit dotaz na dva:

```
// poddotaz
var subquery =
      from x in db.BI_VAZ_Dim_Meas
      where x.id_mea == this.measure
      select x.id_dim;
// hlavni dotaz
  var categories =
     from a in db.BI_Categories
     from b in db.Languages
     from c in db.BI_Trans_Cats
     from d in db.BI_VAZ_Cat_Dims
     from e in db.BI_VAZ_Dim_Meas
     where a.id_cat == c.id_cat && d.id_cat == a.id_cat && b.@short == user_lang && b.
          id_lang == c.id_lang
              && subquery.Contains(e.id_dim) && e.id_dim == d.id_dim
     select new { c.id_cat, c.translated };

  //odstrani duplicity
```

```
categories = categories.Distinct();
```

Výpis 31: Zjištění kategorií obsahující dimenze v závislosti na faktu - LINQ

Situace v LINQ je zde podobná jako v SQL. Poddotaz zde odpovídá poddotazu v příkladu SQL výše. Stejně hlavní dotaz odpovídá SQL. Rozdíl je ve způsobu zápisu (dva oddělené dotazy) a ve způsobu vyhodnocení na straně serveru (provádí dva dotazy). Jak bylo vidět výše, poddotaz v SQL se připojuje klíčovým slovem IN. V případě LINQ je zápis: *poddotaz.Contains(Tabulka_hlavni_dotaz.Atribut_hlavni_dotaz)* - v tomto případě pak *subquery.Contains(BI_VAZ_Dim_Meas.id_dim)*. Zde metoda *Contains()* nahrazuje IN z SQL. I zde bylo po provedení obou dotazů odtranit duplicitní kategorie.



Obrázek 16: Závislosti dimenzí na measures a na kategoriích

Pro lepší názornost bylo na obrázku 16 naznačeno, které komponenty se návzájem ovliňují a to včetně kroků uživatele. Prvním krokem je výběr z neprázdných kategorií. Podle kategorie je možno vybrat měřítka (fakty) v dané kategorie. Druhým krokem je nastavení dimenze/dimenzí (čili user control). Začíná se výběrem kategorie. Dle vybrané kategorie a měřítkalze vybrat dimenzi. Posledním nepovinným krokem je výběr filtru dimenze. Dimenze jsou pro zobrazení výsledku nepovinné. Na obrázku je modře označena user control. Vyskytuje se zde dvakrát, jednou pro dimenzi kategorie sklad, po druhé pro kategorii karty.

5.2.2 Zobrazení výsledku z datové kostky

Nyní uživatel přes webové rozhraní vytvořil MDX dotaz a ten je potřeba zobrazit. Zobrazení se provádí do nového okna. Uživatel má tedy možnost zadání dotazu lehce změnit v původním okně. Druhou výhodou zobrazení v novém okně je možnost otevření několika výsledků dotazů najednou - každé ve svém okně. Jako základní zobrazení bylo zvoleno zobrazení do tabulky. Opět byly využity Telerik komponenty. Místo standardní komponenty GridView byl použit RadGrid od zmíněné firmy. Výhodou jsou například možnost

stránkování, přesouvání sloupců atd. Načtení dat z kostky bylo provedeno pomocí technologie ADOMD.NET.

```
        AdomdConnection conn = new AdomdConnection();
    conn.ConnectionString = this.connectionstring;
        conn.Open();
    AdomdCommand cmd = new AdomdCommand();
    cmd.Connection = conn;
    cmd.CommandText = query;

    AdomdDataAdapter da = new AdomdDataAdapter(cmd);
    DataSet ds = new DataSet();
    da. Fill (ds);
    data.set_DataSet(ds);
    RadGrid1.DataSource = data.get_DataTable();
    RadGrid1.DataBind();
```

Výpis 32: Načtení dat z kostky

Základem je vytvořit připojení na kostku. Před otevřením spojení je třeba nastavit connection string (textový řetezec obsahujicí informace pro připojení k serveru - IP, přihlášení apod). Po otevření spojení byl vytvořen příkaz pomocí objektu *AdomdCommand*. Tomu to objektu byl krom spojení přiřazen MDX dotaz vygenerovaný uživatelem. V dalším kroku bylo třeba načíst výsledek. Pomocí objektu *DataAdapter* byl výsledek načten a uložen do *Dataset*-u. Dataset musel být přepočítat kvůli správne funkčnosti. Nakonec byl přepočítaný dataset ve formě tabulky připojen k RadGrid komponentě a jako vždy zavolána metoda *Databind()*.

Přepočet DataSetu pak vypadá takto:

```
    DataSet dsNew = new DataSet();
dsNew.Tables.Add(new DataTable("FromCube"));

foreach (DataColumn col in ds.Tables[0].Columns)
{
    DataColumn newCol = new DataColumn();
    newCol.ColumnName = col.ColumnName;
        // rozpoznani measures...
    if ((col.ColumnName).Contains("Measures") || (col.ColumnName).Contains("&"))
    {
        newCol.DataType = typeof(System.Double);
    }
    else
    {
      // dimenze, takze datovy typ podle typu dat
      string s= DimDataType(col.ColumnName);
      if (s == "int") { newCol.DataType = typeof(int); }
      if (s == "double") { newCol.DataType = typeof(int); }
     else if (s == "string") { newCol.DataType = typeof(string); }
     else if (s == "date") { newCol.DataType = typeof(DateTime); }
     else { newCol.DataType = col.DataType; }
    }

    dsNew.Tables[0].Columns.Add(newCol);
```

```
}

foreach (DataRow row in ds.Tables[0].Rows)
{
    dsNew.Tables[0].Rows.Add(row.ItemArray);
}


// do prazdych bunek 0 (jinak nefuguje filtrace gridu)
foreach (DataColumn column in dsNew.Tables[0].Columns)
{
    foreach (DataRow row in dsNew.Tables[0].Rows)
    {
        if (row[column].ToString().Length < 1)
        {
            row[column] = 0;
        }
    }
}
```

Výpis 33: Přepočet DataSetu

Je třeba projít všechny sloupce a nastavit jim datový typ a data přidat do tabulky. U faktu víme, že se jedná o číslo (nastaven datový typ float). U dimenzí nastavíme datový typ podle Bez tohoto přepočítání se výsledek zobrazil chybně. Jakmile jsou nastaveny datové typu, procházim dataset znovu a do prázdných (null) buněk dávám nuly. Datová kostka občas vrací prázdné řádky a Grid měl pak problém s třízením a filtrováním. Po dosazení nuly za chybějící hodnotu faktů (measures) je vše v pořádku a funkční.

Uživatel si může nechat data zobrazit do grafu. Zde se počítá s jistou inteligencí uživatele - předpoklad je, že si bude vědet rady s výběrem správného typu grafu. Přidal jsem základní typy grafu (případné rozšíření není problém - otázka pír minut úprav v kódu):

- Bars (sloupcový)
- Areas (plošný)
- Lines (spojnicový)
- Pie (koláčový)
- Point (bodový)
- Spline Area (plošný)
- Stacked Bar
- Stacked Spline Area

Názvy grafů byly ponechány v angličtině. Později podle požadavků uživatelů bude možná změna na češtinu. Případné rozšíření by mohlo být nastavit názvy grafu podle jazyka uživatele v ERP systému.

Zobrazení grafu je otázkou několika řádků kódu. Samotné přidání hodnot do grafu je otázkou dané komponenty, zde je zdrojový kód číslo 34 výmluvný sám za sebe. Uvedeme zjistění hodnot, které je třeba zobrazit v grafu. Zde si uživatel přeje zobrazit fakty (measures) jako hodnoty a dimenze jako osy. Zjistění faktů probíhá podle datového typu. Pokud je datový typ sloupce double (desetinné číslo), pak se jedná o measure (viz nastavení datových typů při přepočítání DataSet-u výše). Ostatní položky jsou dimenze.

```
foreach (DataColumn column in tab.Columns)
        {
            if (column.DataType == typeof(double))
            {
                ChartSeries chartSeries = new ChartSeries();
                chartSeries.Name = column.ColumnName;
                chartSeries.Type = TypGrafu(CB_charts.SelectedValue);
                RadChart2.Series.Add(chartSeries);
                foreach (DataRow row in tab.Rows)
                {
                    // kolacovy typ ma vlastni zpubos pridavani dat
                    if (TypGrafu(CB_charts.SelectedValue) == Telerik.Charting.
                        ChartSeriesType.Pie)
                    {
                        ChartSeriesItem a = new ChartSeriesItem(Double.Parse(row[column].
                            ToString()));
                        a.Name = row[column].ToString();
                        chartSeries.AddItem(a);
                    }
                    else // ostatni uz jednotne
                    {
                        chartSeries.AddItem(Double.Parse(row[column].ToString()));
                    }
                }
            }
        }
        // a zviditelnit graf
        RadChart2.Visible = true;
```

Výpis 34: Přidání hodnot do grafu

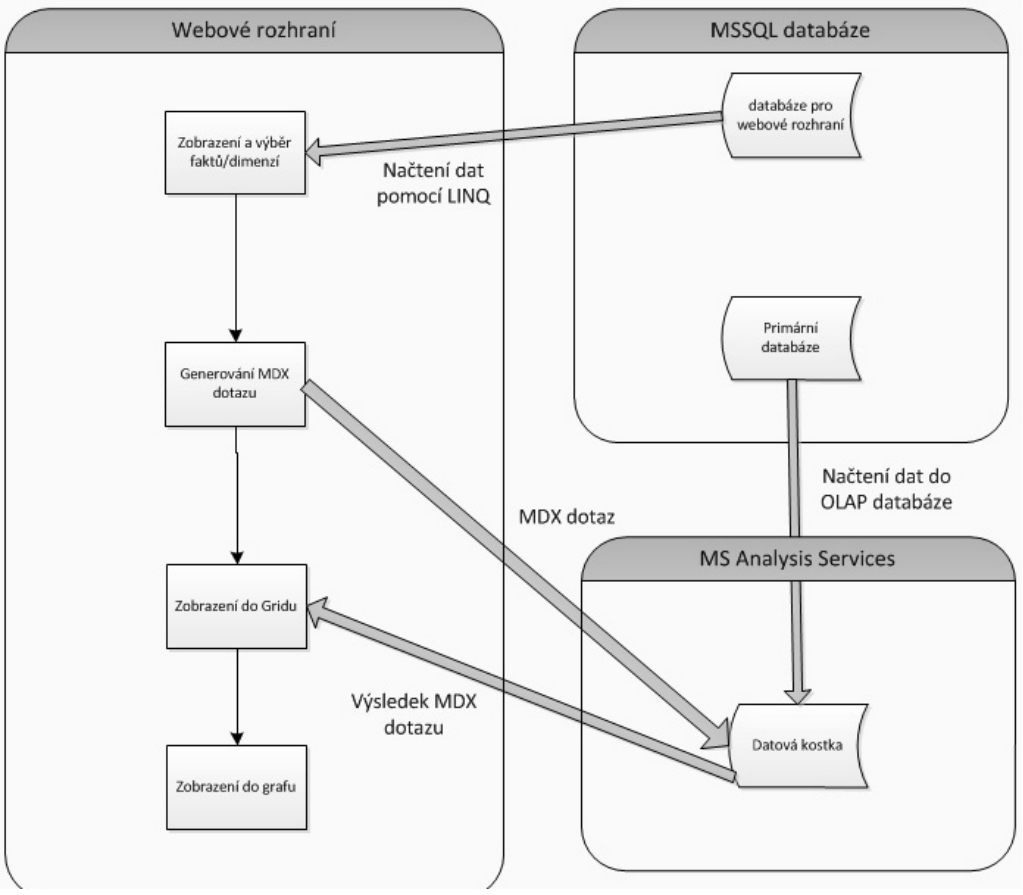
Uživatel si může zvolit popis os. Načtení dimenzí pro osu X se provádí následujícím kódem:

```
CB_X_Axis.Items.Clear();

// prazdna (vychozi) polozka
RadComboBoxItem a0 = new RadComboBoxItem();
a0.Text = "";
a0.Value = "100";
a0.Enabled = false;
a0.Selected = true;
CB_X_Axis.Items.Add(a0);

GridColumn[] col = RadGrid1.MasterTableView.AutoGeneratedColumns;
// projdeme vsechny sloupce (dimenze)
```

```
for (int i = 0; i < col.Count(); i++)
{
    RadComboBoxItem a = new RadComboBoxItem();
    a.Text = col[i].UniqueName;
    a.Value = col[i].UniqueName;
    CB_X_Axis.Items.Add(a);
}
```

Výpis 35: Načtení dimenzí pro osu X

V prvním kroku byly vymazány případné položky v komponentě. Poté byla pomocí *Items.Add()* přidána prázdná (výchozí) položka. Všechny sloupce (dimenze) z tabulky byly načteny pomocí *RadGrid1.MasterTableView.AutoGeneratedColumns*. Ve zbývajícím kroku projdeme pomocí cyklu všechny názvy dimenzí a zapsány do komponenty. Změna dimenzí nemá vliv na samotné zobrazení grafu - mění se čiste popis osy X. Nicméně zvyšuje se přehlednost pro uživatele hlavně pak v použití grafu jako obrázku například v dokumentech firmy. Export grafu do obrázku není součástí řešení práce, ale je to jedna z možností případného rozšíření na žádost uživatele webového rozhraní.



Obrázek 17: Dataflow webového rozhraní

Na závěr shrneme dataflow webového rozhraní. Jak je vidět na obrázku číslo 17, v prvním kroku má uživatel možnost vygenerovat MDX dotaz podle vybraného faktu a dimenzí. Fakty a dimenze se načítají z MS SQL databáze pomocí technologie LINQ a podrobně jsem tento proces rozepsal výše. Po potvrzení volby uživatelem dojde k vygenerování dotazu. Dotaz se odešle do OLAP databáze. Výsledek se otevírá do nového okna, kde se zobrazí grid (tabulka) s požadovanými daty. Data jsou načtena jako výsledek MDX dotazu do DataSetu a zobrazena v Gridu. Volitelně si může nechat uživatel zobrazit graf. Ten data už nezískává znovu z datové kostky, ale ze stejného DataSetu, který byl použit pro Grid.

6 Závěr

Hlavní cíl práce, a to vytvoření nového modulu pro stávající ERP systém, byl splněn. ERP systém obsahuje základní a fungující modul, který je v době psaní této práce ve fázi testování u klienta. Základní požadavky, a to hlavně zobrazení relevantních hodnot, splňuje. Oproti zadání se z modulu pro marketing nakonec stal modul pro komplexní analytiky, což hodnotím jako splnění požadavku na lepší úrovni, než byly původní představy firmy. Modul je lehce rozšiřitelný, a to díky použití Business Intelligence a technologie Microsoft SQL Server Analysis Services. Datová kostka poskytuje snadnou správu dat a případnou rozšiřitelnost o další dimenze a fakty. A to vše při menších nárocích na hardware než při použití jiného řešení (například generátorů SQL dotazů).

Webové rozhraní je pro uživatele po krátkem zaškolení v rámci představení celého ERP systému snadno pochopitelné. Přidání nových hodnot do webového rozhraní je otázkou přidání několika málo záznamů do databáze, čili i zde se povedlo uhlídat snadnou rozšiřitelnost. Samotné zobrazení dat je pak otázkou výpisu dat do tabulky. Vzhledem k použití Telerik komponent má uživatel pokročilejší možnosti jako například filtraci dat v tabulce. Volitelnou možností je i zobrazení dat do několika typů grafů.

Budoucí rozšíření očekávám hlavně v části webového rozhraní, a to například přidání šablon na uložení nastavení dimenzí. V zobrazovací části pak například export dat z tabulky do různých formátů (Excel tabulka, PDF) apod. Tato rozšíření půjde provést snadno, šlo by o přidání jen několika komponent a metod. Samotné načítání dat z kostky je vyřešeno dobře a neočekávám zde změny.

Zdrojové kódy jsou neveřejné - jedná se o firemní know-how. Webové rozhraní je v současné době dostupné na adrese http://bi.dev.xevos.cz/VSB.

Jan Bauch

7 Literatura

[1] Alberto Ferrari, Chris Webb, Marco Russo *Expert Cube Development with Microsoft SQL Server 2008 Analysis Services*, Birmingham,: Packt Publishing Ltd., 2009.

[2] Mike Hotek *Microsoft® SQL Server® 2008 Step by Step*, Microsoft Press: Microsoft Press, 2008.

[3] Petra Hýžová *Uplatnění principů Business Intelligence v praxi*, Brno: Masarykova univerzita - Magisterská diplomová práce, 2006.

[4] Jana Šarmanová *Informační systémy a datové sklady*, Ostrava: VŠB-TUO, 2007.

[5] Vincent Rainardi *Building a Data Warehouse: With Examples in SQL Server*, New York: Apress, 2008.

[6] Paolo Pialorsi, Marco Russo *Microsoft LINQ Kompletní pruvodce programátora*, Brno: Computer Press, 2009.

[7] Bill Evjen, Scott Hanselman, Devon Rader *ASP.NET 3.5 v jazycích C# a Visual Basic Programujeme profesionálne*, Brno: Computer Press, 2009.

[8] Robert E.Walters, Michael Coles, Robert Rae, Fabio Ferracchiati, Donald Farmer *Mistrovství v Microsoft SQL Server 2008*, Brno: Computer Press, 2009.

[9] Luboslav Lacko *Business Intelligence v SQL Serveru 2008 - Reportovací, analytické a další datové služby* , Brno: Computer Press, 2009.

[10] John Sharp *Microsoft Visual C# 2008 - Krok za krokem* , Brno: Computer Press, 2009.

[11] *Developer Productivity, Team Productivity, Automated Testing Tools, Web Content Management*, URL: <http://www.telerik.com/> [cit. 2010-04-01].

[12] *Data Mining*, URL: <http://datamining.xf.cz/> [cit. 2010-04-01].

[13] *ADOMD.NET*,
URL: <http://msdn.microsoft.com/en-us/library/ms123483.aspx/> [cit. 2010-04-01].

[14] *LINQ: .NET Language-Integrated Query*,
URL: <http://msdn.microsoft.com/en-us/library/bb308959.aspx/> [cit. 2010-04-01].

A Uživatelská příručka

V této části sepíšeme základní "manuál" pro ovládání webového rozhraní. Zde by bylo vhodné upozornit na omezení ohledně prohlížečů. Mezi doporučené patří Internet Explorer verze 8 nebo Firefox verze 3.5. Uživatel si v prvním kroku vybere kategorii měřítka (fakty/measures), tedy hodnoty, se kterými chce počítat. Každá kategorie obsahuje nejméně jednu položku. Položky, které počítají s procenty jsou pro přehlednost označeny pomocí značky [%]. Vzhledem k použití modulu pouze vybranými lidmi, kteří projdou zaškolením, není zde potřeba vysvětlit všechny hodnoty a dimenze. Ty jsou vytvořeny právě na základě požadavků klienta, který proto přesně ví, o co jde. Po výběru měřítka je možnost nechat hodnotu zobrazit. V tomto případě bude bez filtrů výsledkem vždy pouze jedno číslo.

Vzhledem k tomu, že takový výsledek by uživateli ve většině případů nepomohl, je zde možnost danou hodnotu rozdělit (filtrovat/agregovat) pomocí dimenzí. Dimenze se nacházejí opět v kategoriích. I zde platí, že každá kategorie obsahuje alespoň jednu dimenzi (přesněji pak atribut z dimenze). Podle typu atributu je možno nastavit filtr hodnot. Pokud jde o číslo, máme k dispozici filtry jako větší, menší, rovno apod. V případě textu jde například o "obsahuje" a o "neobsahuje". Tímto omezením lze zobrazit jen určité výrobky apod. Dále je zde možnost volby, zda-li se filtr zobrazí do sloupce nebo řádku. Tato volba ovlivňuje kromě samotného vzhledu tabulky i výsledné zobrazení grafu.

Po nastavení potřebných filtrů se stisknutím příslusného tlačítka zobrazí do nového okna výsledek. Zde upozorníme na případ, kdy je otevírání nových oken v prohlížeči zakázáno. Webové rozhraní se pak zdá nefunkční (po stiknutí tlačítka se nic nestane). V takovém to případě je třeba povolit otevírání nových oken v prohlížeči. Takový způsob zobrazení je zvolen proto, aby se nastavení filtrů zachovalo. Případná jejich úprava je pak snadnější. Dalším důvodem pro toto řešení je možnost otevřít si více výsledků najednou.

Zobrazení může podle náročnosti dotazu trvat i několik vteřin. V tomto případě je trpělivost na místě, tato situace by se měla vyskytovat však jen velmi sporadicky. Výsledek se pak zobrazí do tabulky. Tuto je pak možno filtrovat, přesouvat sloupce apod. Dále je zde možnost nechat si zobrazit data do grafu. Zobrazení se provede volbou typu grafu a to:

- Bars (sloupcový)
- Areas (plošný)
- Lines (spojnicový)
- Pie (koláčový)
- Point (bodový)
- Spline Area (plošný)
- Stacked Bar

- Stacked Spline Area

Po výběru typu grafu se zobrazí samotný graf. Webové rozhraní automaticky rozpozná, které hodnoty v grafu zobrazit (automaticky vybere měřítka). V případě potřeby použití grafu jako obrázku, například do prezentace je možnost změnit popis osy X ”rolovátkem” na pravé straně.

B Obsah CD

- text práce